APPLICATION FOR A UNITED STATES PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE

(Case No. 00-625-F)

Title: **A System and Method for Providing and Displaying Information Content**

Inventors:

Simon Hunt
Kevin Wallace
Bruce Wiatrak
Igor Matlin
Thomas Hayosh
Sean Patterson
Brent Dafforn
Cecile McHugh

Assignee: Novarra, Inc.
3232 N. Kennicott
Arlington Heights, IL 60004

REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of U.S. patent application Serial No. 10/280,263, titled "System and Method for Displaying Information Content With Selected Horizontal Scrolling" and filed October 25, 2002, which is incorporated herein by reference in its entirety, and which in turn is a continuation-in-part of U.S. patent application Serial No. 09/842,474, titled "System and Method for Accessing Information Content" and filed on April 25, 2001, which is incorporated herein by reference in its entirety, and which in turn claims priority to United States provisional application Serial No. 60/199,858 filed on April 26, 2000, which is also incorporated herein by reference in its entirety. U.S. patent application Serial No. 10/280,263 is also a continuation-in-part of U.S. patent application Serial No. 09/843,036, titled "System and Method for Adapting Information Content for an Electronic Device" and filed on April 25, 2001, which is incorporated herein by reference in its entirety, and which in turn claims priority to U.S. provisional application Serial No. 60/199,858 filed on April 26, 2000.

FIELD OF THE INVENTION

This present invention relates generally to the field of network communications. More specifically, it relates to a system and method for accessing, adapting, and presenting information content for electronic devices.

BACKGROUND OF THE INVENTION

Today, an abundant amount of meaningful and feature rich information content is truly at one's fingertips. Currently, using a personal computer (PC) and PC-based browser, one can find information online regarding just about anything they desire. One can communicate with people on the other side of the U.S. or world, set up a teleconference call, tap into the resources of other computers across the earth, search through the world's finest libraries, and visit images from the world's most remarkable museums. One can even use the PC-based browser to watch videos and listen to their favorite music, monitor the financial markets, find the local weather forecast, go shopping, download application software, and so on. Currently, all of this can be done with a personal computer and a PC-based browser that is tapped into a feature rich network of computers such as the Internet, Intranet, or Extranet.

At the same time, the field of communications, and more specifically wireless telecommunications, is currently undergoing a radical expansion. This technological expansion allows an electronic device, such as mobile personal digital assistant (PDA), cellular phone, pager, and other electronic devices to connect to the same information sources, such as a web server or database, as one could with the PC and a PC-based browser.

Unfortunately, this feature rich information content was developed for a standard PC-based browser, not a mobile or portable electronic device that might be limited in memory, screen size, bandwidth, navigation capabilities, power consumption, processing power, etc. For example, an electronic device, such as portable PDA, with a small screen

McDonnell Boehnen Hulbert & Berghoff
300 S. Wacker Drive, Floor 3200
Chicago, IL 60606
Telephone: 312-913-0001

size may be inappropriate to display the same information content originally intended for a PC-based browser, and viewed on a 15-inch or greater size display monitor. Consequently, the PDA cannot be able to faithfully access and display information content as it was originally intended to be viewed. Therefore, it would be desirable to access, organize, and navigate information content including applications.

In another example, a mobile or wireless device with only low bandwidth capability may be unable to view information content intended for only high bandwidth applications. Parameters such as the mobile or wireless device's network connection, memory capacity, power restrictions, or other limitations of the device may require customization of information content that is delivered to or from the device. Therefore, it would be desirable to streamline the information content such that the desired content is received and presented at the mobile device as it was intended to be viewed.

In yet another example, current electronic devices cannot take full advantage of dynamically generated content and interactive Web sites that are typically hosted on today's Web servers. According to this example, scripting languages like JavaScript or Jscript allow a user on a PC-based browser to interact with markup language such as Hypertext Markup Language (HTML) source code, thus enabling the use of dynamic content. However, it would be desirable for a portable electronic device, although possibly having limited abilities, to also utilize the modern and current scripting languages.

Currently, information content is sent to the device, but often in a format that the appliance, user, or network cannot conveniently accommodate, which produces

McDonnell Boehnen Hulbert & Berghoff
300 S. Wacker Drive, Floor 3200
Chicago, IL 60606
Telephone: 312-913-0001

undesirable results. For example, the data content might be unreadable on the display, displayed in an unorganized fashion, be too voluminous or bandwidth intensive to be received or displayed, and so on.

Creators of content may use tables and/or frames to control the placement of images and text and take better advantage of the available display screen. For further effects, tables and/or frames can also be nested within tables. Tables providing content designed for larger desktop screen displays, however, may not fit on smaller screen displays, such as on a personal digital assistant, cell phone, pager or internet information appliances, or even when windows are resized. For example, content may be wider than the available screen display with portions of the content then be rendered outside the viewable area of the display. The content is then only partially viewable with portions chopped off, which is at a minimum aesthetically displeasing and may also reduce the readability and usability of the content when significant portions are cut off.

To view the content information outside the limited viewable area, a user scroll request such as horizontal scrolling across the content page may be necessary. By horizontally scrolling, the user can selectively show the hidden portions of the content information. Scrolling, however, requires additional input from the user, which is inconvenient, and still does not allow the entire page to be seen at once. Scrolling across the page to see the hidden content usually hides the portion of the page that was original viewable and the entire content is still not viewable. Horizontal scrolling can be eliminated by reducing the width of the information content to always fit within the width of the screen. Indiscriminately reducing the width of information content destroys the

McDonnell Boehnen Hulbert & Berghoff
300 S. Wacker Drive, Floor 3200
Chicago, IL 60606
Telephone: 312-913-0001

integrity of the content as the creator intended and possibly results in the loss of information.

It would be desirable to present content on smaller screen displays to preserve the content, yet selectively minimize the scrolling necessary to view content.

McDonnell Boehnen Hulbert & Berghoff
300 S. Wacker Drive, Floor 3200
Chicago, IL 60606
Telephone: 312-913-0001

SUMMARY OF THE INVENTION

A system and method is provided that enables electronic devices with limited hardware or network capability to successfully access the same feature rich information content as full featured PC-based browsers with a large display screen, extensive user input facilities (e.g., mouse, keyboard, etc), high CPU power, large memory, reliable network connections, a reliable power supply, and so on.

According to a preferred embodiment, the information content is optimized to selectively minimize the horizontal scrolling required to view the content. The need for horizontal scrolling is selectively removed where not necessary to preserve the context. Each component or sub-component of the original area, such as a frame, table row, table cell or nested table, is considered separately and may be preserved, resized, or replaced. Where the context requires that the content be wider than the viewable area of the screen, horizontal scrolling is preserved.

In an aspect of the present embodiment, the system enables an electronic device to access a number of different information sources including, but not limited to, marked up content like HTML, XML, WML, voice and multimedia. In the exemplary embodiment, a script execution engine is utilized to support scripting technologies such as JavaScript that dynamically generate content.

According to another aspect of the present embodiment, a distributed browser includes separable components, a server browser and a client browser, that enable an electronic device with a small display to efficiently access information content. In the exemplary embodiment, the server browser and the client browser work together to

access the information content by separating functionality between the browsers, irrespective of the component's location. Preferably, the functionality applied to optimize information content access, arrangement, transmission, and navigation can be performed by the server browser rather than the client browser hosted on the portable or mobile device.

According to another aspect of the present embodiment, a QDOM converts data content into a document object tree represented by a mutable object having an array structure. Based on the nodes of the object tree, the QDOM generates an array of primitive data types for efficiently developing an optimized standard structure for use by a normalizer or other processing modules. In the manner, the QDOM extends the World Wide Web Consortium (W3C) DOM interface definition to an efficient model that provides high speed parsing, storage, and access while minimizing memory resource requirements.

In another aspect of the present embodiment, a normalizer adaptively tailors and folderizes markup based information content to accommodate an electronic device's particular software, hardware, and network characteristics. In the exemplary embodiment, the normalizer organizes any markup based information content into folders of interest. The user of the electronic device can then further explore the folders of interest as desired.

In another aspect of the present embodiment, a normalizer is utilized to selectively minimize the amount of horizontal scrolling required by the end user. In the exemplary embodiment, the normalizer modifies layout constructs such as tables and

frames to achieve the reduction in horizontal scrolling. Nested layout is flattened and objects that are wider than the electronic device screen such as tables or images are analyzed to determine if they can be reduced to fit the screen without affecting context.

In yet another aspect of the present embodiment, metatags embedded in a markup language at the information source can provide instructions to the normalizer to take appropriate actions. Use of metatags can allow customization of original information content if a modified outcome is desired at the electronic device. In the exemplary embodiment, the metatags provide instruction to an automatic normalizer including, but not limited to, direct output of information content without normalization, the promotion of content into or out of folders, and dropping or filtering information content from the serialized output to an electronic device.

In another aspect of the present embodiment, pattern-matching templates are utilized to normalize the presentation of accessed information content. In the exemplary embodiment, a template normalizer utilizes regular expression pattern-matching to impose a template over a document and attempts to match the template to the document.

In another aspect of the present invention, an event translator provides additional compatibility with commercially available client browsers or end user applications that employ standardized protocols. In the exemplary embodiment, the event translator can be utilized on the server browser or the client browser to provide compatibility with standard client browsers.

In an aspect of the present embodiment, a serializer dynamically formats normalized content to a form that is optimized for a particular electronic device. The

serialized output can be formatted to suit industry standard browsers, or targeted to an electronic device using the client side browser.

The present embodiments allow for electronic devices with limited hardware capability to access, on the fly, feature rich static and dynamic content, and applications. The server browser enables a client browser that utilizes a particular markup language to access information content that is of any type of markup language or technology. The distributed browser minimizes the functionality required on the device and implements the CPU and memory intensive functions on a server in the network, thus allowing wireless devices, with intermittent, limited connectivity, processing power capability etc. to provide a similar experience achieved with a desktop PC.

In an aspect of the invention, the system supports tiers of devices. For devices with enough processing power and capability, the system supports a mode where the client browser is able to access and render content from the information source without the need for a separate server browser. In this scenario, the server browser components are essentially co-existent with the client browser components.

In another aspect of the invention, the client browser can optionally utilize the server browser as a means to enhance capabilities, improve speed or add function. The use of the server browser by the client browser can be initiated either manually via a user preference or automatically via pre-defined algorithms that take into account the hardware and software capabilities of the device and the characteristics of the wireless network used for the request/response as well as the application needs indicated by the information source. Multiple components including a serializer, normalizer, client

McDonnell Boehnen Hulbert & Berghoff
300 S. Wacker Drive, Floor 3200
Chicago, IL 60606
Telephone: 312-913-0001

browser, and/or the event translator work in conjunction with each other to convert user events within one markup domain into another markup domain while staying in the transaction to translate the meaning of the interaction appropriately. Thus, for example, user events such as scrolling, clicking, voice commands interact with the QDOM to result in a change in presentation of the content.

In another aspect of the present invention, content can be rendered based on the particular device. For example, frame handling features can be implemented to dynamically expand frames so as to reduce or eliminate scrolling within frames or to detect use of a main frame versus navigation bars and banners. Auto-detection of the appropriate view mode and multiple window support might additionally be implemented. Also, adaptive rendering, such as selective image rendering and ad blocking features might be implemented.

In yet another aspect, network optimization might be implemented to send content in a form best suited to the performance characteristics of a user's particular network and device. Also, feature might be implemented that provide for carrying data between forms, thereby allowing the data to be available even if network connectivity is lost.

Additionally, the present embodiments provide significantly higher speed and an efficient use of network bandwidth as desired information content can be cached on the server browser and on the client browser, if so desired, to enable quick access to the desired portions of the information content.

The present embodiments also provide for server browser-centric access to user profile and client browser state information (such as cookies), thereby facilitating the use of multiple devices by a single user.

The present embodiments provide a number of advantages and applications as will be more apparent to those skilled in the art. The exemplary embodiments utilize distributed architecture for adaptively tailoring information content to electronic device's hardware and network characteristics.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a high-level diagram illustrating an exemplary system for accessing, adapting, and presenting information content to electronic devices;

FIG. 2 is a diagram further illustrating the server browser of FIG. 1;

FIG. 3 is a diagram further illustrating the event translator of FIG. 1;

FIG. 4 is a diagram further illustrating the client browser of FIG. 1;

FIG. 5 is a diagram illustrating an events message utilized in the exemplary system of FIG. 1;

FIG. 6 is a diagram illustrating a transmission ACK/NAK message utilized in the exemplary system of FIG. 1;

FIG. 7 is a diagram illustrating a security handshake request/response and ACK/NAK message utilized in the exemplary system of FIG. 1;

FIG. 8 is a diagram illustrating an exemplary process of the QDOM of FIG. 1;

FIG. 9 is a diagram further illustrating the process of the QDOM of FIG. 8;

FIG. 10 is a diagram illustrating the process of the normalizer of FIG. 1;

FIG. 11 is a diagram further illustrating the process of FIG. 10;

FIG. 12 is a diagram showing an exemplary conversion of output from the system of FIG 1;

FIG. 13 is a flowchart of an exemplary embodiment implementing processing for selective horizontal scrolling;

FIG. 14 is a flowchart of an exemplary process for nesting that can be used in conjunction with the flowchart of FIG. 13 to implement selective horizontal scrolling;

McDonnell Boehnen Hulbert & Berghoff
300 S. Wacker Drive, Floor 3200
Chicago, IL 60606
Telephone: 312-913-0001

FIG 15. is a flowchart of an exemplary process for table size calculations that can be used in conjunction with the flowchart of FIG 13. to implement selective horizontal scrolling;

FIG. 16 is an example of original content for a typical screen display and how selective horizontal scrolling reformats content for a small screen device;

FIG. 17 shows selective horizontal scrolling exemplary handling of frames;

FIG. 18 shows an exemplary handling of images in horizontal scrolling mode;

FIG. 19 shows an exemplary handling of popup windows.

FIG. 20 shows an further exemplary handling of popup windows;

FIG. 21 shows an exemplary reduction of content layout of common navigation areas;

FIG. 22 shows an exemplary handling of absolute sized frames;

FIG. 23 shows an exemplary handling of relative sized frames;

FIG. 24 is a flowchart of an exemplary process for calculating frame sizes so that scrolling within frames is not required;

FIG. 25 is a flowchart of an exemplary process for calculating initial column widths for nested frames that can be used in conjunction with FIG. 24 to implement frame expansion so that scrolling within frames is not required;

FIG. 26 is a flowchart of an exemplary process for calculating minimum frame widths that can be used in conjunction with FIG. 24 to implement frame expansion so that scrolling within frames is not required;

FIG. 27 is a flowchart of an exemplary process for calculating actual column widths for frames that can be used in conjunction with FIG. 24 to implement frame expansion so that scrolling within frames is not required;

FIG. 28 is an example of different layout results in different display modes showing how form input visibility is used to automatically determine the best layout;

FIG. 29 is a flow chart of an exemplary process for deciding when to automatically switch the client browser from server mode to proxyless mode and back again;

FIG. 30 shows an exemplary process for determining when to automatically switch the client browser from server mode to proxyless mode based on network information;

FIG. 31 shows an exemplary process for using statistical history for determining when to automatically navigate links on the server to improve the speed and efficiency of browsing; and

FIG 32 shows an exemplary process for the "follow-me" push service.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

FIG. 1 shows a high-level block diagram illustrating an exemplary system 100 for accessing and adapting feature rich information content for presentation on an electronic device 104. The accessed and adapted information content is transmitted between an information source 102 and the electronic device 104.

The information source 102 includes any type of device such as a web server, application server, database or other backend system, or any interface such as a web service to an information provider. Preferably, the information source 102 provides information content expressed in a markup language, such as those markup languages known in the art including Hypertext Markup Language (HTML), Extensible Markup Language (XML) with or without Extensible Style Sheets (XSL), VoiceXML, Extensible Hypertext Markup Language (XHTML), or Wireless Markup Language (WML). Furthermore, the information content can store images, video, audio information. Preferably, the information source 102 can be accessed through an information access network 106 such as a local area network (LAN) or wide area network (WAN).

The electronic device 104 includes any type of device such as a personal computer (PC), wireless telephone, personal digital assistant (PDA), hand-held computer, network appliance, and a wide variety of other types of electronic devices that might have navigational capability (e.g., keyboard, touch screen, mouse, etc.) and an optional display for viewing downloaded information content. Furthermore, the electronic device 104 can also include a device such as a set-top box, Internet access appliance, infra-red remote control used with a set-top box, and so forth. Moreover, the electronic device 104 can

include any type of device that has the capability to utilize speech synthesis markups such as W3C (www.w3.org) Voice Extensible Markup Language (VoiceXML).

Information content from the information source 102 is preferably retrieved and tailored for use on the electronic device 104 by a distributed browser 108. The distributed browser 108 is generally made up of a server browser 110 and a client browser 112. By utilizing the distributed browser 108, smaller electronic devices with limited hardware capability can access feature rich information or data. Moreover, the distributed browser 108 allows for efficient use of the communications network 114 bandwidth. Of course, electronic devices with high processing power, fast network connection, and large memory can also use the present embodiments.

In the exemplary embodiment, the server browser 110 and the client browser 112 are hosted on separate platforms. For example, the server browser 110 might be hosted on a back-end server, and the client browser 112 might be hosted on the electronic device 104. However, it should be understood that the server browser 110 and client browser 112 can be hosted on the same platform such as on an electronic device, especially if the platform or electronic device has the appropriate hardware and network capabilities.

The server browser 110 can access information content at the information source 102 via the information access network 106. In the exemplary embodiment, the server browser 110 operates as a client of the information source 102. For example, using a known suite of communications protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP), the server browser 110 can issue a Hypertext Transfer Protocol (HTTP) request to the information source 102 over the information access network 106.

By utilizing HTTP requests, such as is known in the art, the server browser 110 can access information content, including applications, static and dynamic content, at the information source 102. Dynamic content can include script codes such as JavaScript, developed by Netscape (www.netscape.com), and Jscript, developed by Microsoft (www.microsoft.com).

Preferably, communications between the client and server browsers 112 and 110, respectively, are via a defined application protocol implemented on top of a wired or wireless transport layer depending on the nature of the electronic device 104 and communications network 114.

Therefore, the communications network 114 might include a wired network such as those that utilize Ethernet or similarly IEEE 802.3 protocols. The communications network 114 might also include a wireless network such as a local area wireless network (LAWN) or wireless local area network (WLAN). Moreover, the communications network 114 might include wireless networks that utilize other known protocols and technologies such as Bluetooth, wireless application protocol (WAP), time division multiple access (TDMA), or code division multiple access (CDMA). Furthermore, the communications network 114 is not limited to terrestrial networks, but can utilize other forms of transmission, as is known in the art, such as a satellite connection.

To provide an exemplary illustration, assume that a PDA hosts a client browser, a PC hosts a server browser, and the PDA and PC are both connected to an Ethernet network. Then, the client browser and the server browser could perform information transactions over the Ethernet network. Such transactions would utilize Ethernet or

similarly IEEE 802.3 protocols. Nevertheless, in this example, the client and server browsers communicate over a wired network.

In another example, assume that an internet-enabled refrigerator hosts a client browser, a set-top box hosts a server browser, and both could perform information transactions over a Bluetooth or IEEE 802.11 wireless LAN. Then, according to this example, the client and server browsers are communicating over a wireless network.

The distributed browser supports network optimization between the client browser 112 and server browser 110 through the use of a transport layer protocol that effectively increases the speed of transmission of content between these two end-points. This results in reduced latency for the client browser 112 to retrieve content from the information source 102 and hence an overall improvement in the user's experience. This type of optimization is most beneficial when the communication path between the client browser 112 and server browser 110 is over a wireless network.

One implementation of this type of transport layer optimization is based on the Stream Control Transmission Protocol (SCTP) [Network Working Group RFC 2960] with parameters and message sequencing optimized for wireless transmission media. The key feature of the SCTP-based network optimization protocol is support for multiple independent streams of application messages over a single connection. This reduces or can eliminate the "head-of-line-blocking" problem associated with the use of TCP due to the necessity to maintain strict sequence messaging. The ability to support multiple independent streams is key to the application of this protocol to the distributed browser 108 since two or more of the various objects which constitute the content obtained from

the information source 102 can be simultaneously transferred via separate streams over the same connection. Flow control and congestion control are mechanisms utilized by the protocol to maintain the efficiency of the streams.

Another feature of the SCTP-based network optimization protocol is the capability to monitor the reachability of the remote end-point thereby providing transport-level fault tolerance. The SCTP-based network optimization protocol also supports preservation of application message boundaries for logical chunks of data sent over a single stream. This is a particularly efficient method for the client browser 112 to receive independent parts of the same document useful for the support of in-line expansion of folderized content. Lastly, the protocol supports unordered reliable message delivery which provides for the out-of-band messaging capability used to support the server to device push of content, configuration updates and application updates described elsewhere in this document.

In another aspect of the exemplary embodiment, the server browser can also select the most appropriate content for the requesting or target client browser component. The limited capabilities of the hardware, operating system and/or software on the target device may limit the formats for image, audio, video, etc. that can be supported directly. In such cases, it is normal for a browser to report to the information source which particular formats can be supported. One particular browser may support the PNG image format, while another may also support JPEG and GIF formats and yet another may support other, non standard formats (e.g. Palm OS devices support only their own native image formats).

The architecture of the exemplary embodiment allows for the server browser component to translate the original content into formats that are directly supported by the client browser component and/or target device. In addition to this, however, the server browser component can use performance characteristics of the device and network to determine the most efficient format to send. For example, the server browser might take into account the time that might be required for the following actions:

- Converting between formats on the server
- Transmitting of data over the air
- Converting between formats on the client
- Rendering/display/playing of the data by the client.

This is not an exhaustive list, and the server browser might take into account other factors as well.

Based on these factors, the server browser might determine that it is likely more efficient to send the data in a format that while larger, takes less time to convert and render on the device. For example, when communicating with the client on an older Palm OS device using a slow processor and only supporting the native Palm OS bitmap format, it might be more efficient to covert all images to the native Palm OS format. While this format is often far less efficient than the original JPEG format of many images and therefore might take longer to transmit the larger images over the air, the extra transmission time might be more than compensated for by the immediate availability of the image in the native format on the device. On newer Palm OS devices with faster processors, however, the conversion between the image formats on the target device is far

quicker and the advantages in sending the larger format might be lessened or lost. Knowing the performance characteristics of the machine that it is running on, the network in use and the performance characteristics of the client device combined with the original data format and potential delivery formats, the server browser component can determine the most efficient format for delivery. It is apparent to those of ordinary skill in the art that this same algorithm can apply to a variety of data formats used in the industry by browsers, including image, audio, video, etc.

Referring again to FIG. 1, a commercially available or standard client browser 140 can also be supported. Preferably, an event translator 136 is used to convert a request/response protocol, such as an HTTP request, from the standard client browser 140 (e.g., WML, XHTML, cHTML, etc.) to an event that the server browser 110 recognizes. Preferably, the translation process includes event information, content information, and the context of the event such that transactions between the standard client browser 140 and the information source 102 (e.g. HTML form submission) are preserved. Therefore, by using the event translator 136, the server browser 110 can provide an interface to any end user application with a known protocol. Thus, for example, an electronic device 104 can utilize the client browser 112 or standard client browser 140, or both at the same time, if so desired.

The server browser 110 can be hosted on any platform with sufficient hardware capability for performing tasks by the server browser 110 described herein. Such platforms can include but are certainly not limited to desktop or laptop PCs, servers, computer clusters, or embedded devices. It should also be understood that the server

browser 110 can also be hosted on the electronic device 104, especially if the electronic device has the hardware and network connection capability.

According to the exemplary embodiment, the server browser 110 can authenticate itself with the information source 102, facilitate the request of the information using a protocol acceptable to the information source 102 (e.g., an HTTP request for a web server), provide secure transactions with the information source 102, provide secure transactions with the client browser 112, execute embedded scripts or code segments, and resolve necessary external references (e.g., request HTML frames or script sources) to complete the information model. Preferably, the server browser 110 also includes information content caching and data pre-fetch for performance gain.

Furthermore, the server browser 110 can perform information content transformations or apply device specific style sheets to aid in presentation (e.g., display or voice) and navigation (e.g., keyboard, touch screen, or scrolling), and perform content grouping for electronic devices that accepts data in limited quantities.

To deliver these capabilities, the server browser 110 preferably contains the modules including user agent 110, cookie handler 112, QDOM 116, script executor 120, normalizer 124, serializer 128, and connectivity 132, each described below. A session manager 108 is also included to manage the session between the client browser 112 and the server browser 110. Similarly, the session manager 108 can also manage the session between the standard client browser 24 and the server browser 110.

FIG. 2 is a diagram further illustrating the server browser of FIG. 1. The server browser utilizes a user agent 110 for accessing information at the information source 102.

Preferably, the user agent 110 has the functionality of a traditional PC browser (e.g., Netscape Navigator, Internet Explorer, and so forth) as well as extended functionality, described below, due to the distributed nature of the electronic device 104. To access the appropriate information content at the information source 102, the user agent 110 communicates the requested resource identifier to the information source 102.

For example, the user agent 110 might transmit an HTTP request to a remote web server that hosts yahoo.com. According to this example, the user agent 110 would transmit a resource identifier to request a specific web page or ask the remote web server to perform a database query. The request including the resource identifier is broken into HTTP packets and the packets are sent across the Internet's TCP/IP communications infrastructure to the remote web server. The resource identifier then enables the host computer to locate the requested page at yahoo.com and return the information content to the user agent 110.

In addition to transmitting the resource identifier, the user agent 110 might inform the information source 102 of the client browser 112 type, electronic device 104 capabilities, and user preferences in the request headers and receives information identifying the properties of the data received (such as the content type, length and encoding) in the response headers. The headers that are sent back and forth between the information source 102 and the server browser 110 may also contain one or more cookies stored at the server browser 110 on behalf of the client browser 112.

Preferably, the user agent 110 conforms to the broader industry definition of the term as a component of the server browser 110 that acts on behalf of the electronic device

104 to request information from an information source 102. The requested information content can be from any information source including a web server as described above, but is not restricted to a web server. Other sources of information content might include an email server, Instant Messaging server, database or other storage of information. Additionally, the means through which the user agent 110 communicates with the information source 102 includes the HTTP protocol as described above, but of course, is not limited to that protocol.

Beyond industry conformance, the server browser adds the concept of dynamic user agent 110 functionality by enabling the identity information sent in the request to the information source 102 to vary depending on the type of application being accessed. According to standard industry practice, a content server or information source 102 recognizes the properties of a connected client, and understands how the client browser works. Then, the server sends suitable content to the client so that, for example, the Netscape browser may receive different content from Internet Explorer because they report different user agent capabilities in their requests. The different capabilities may be explicitly stated or implied by client name and version information. These and other existing client browsers in the industry always report the same user agent attributes to every content server with every content request. In the exemplary embodiment, however, the user agent 110 attributes that are contained in the request can vary dynamically between requests. In doing so the identity which extracts the most content from the information source 102 will be represented by the user agent 110. The identity to be represented will be determined via pre-configured rules and/or via algorithms taking into

account the response information obtained from prior requests to the information source 102.

Information content might also use XML information content and XSL style sheets instead of HTML as the preferred internet/intranet information content format. By using XML information content and an XSL style sheet, it can provide a clear separation of data and presentation. The XSL style sheet is applied to the XML information content by an XSLT engine to present the information content to an electronic device 104. The XSL style sheet is applied to the XML information content at the information source 102, but preferably the client browser 112 can also apply the XSL style sheet to the XML information content. In this case, the server browser 110 preferably employs an XSLT engine to apply the XSL style sheet to the XML data before normalizing to produce content. One such example would be a WML client browser used to request an XML + XSL combination that produces XHTML. Alternatively, the information content author may choose to use the original XML and apply templates and/or wireless markup instead of or in combination with XSL style sheets.

In addition to providing normalizer functionality, the system can also use templates and meta-tag markup to alter the original information content to better suit an end user application for which it was not originally designed. This can be achieved through the addition, removal or substitution of sections of content, tags and attributes (separately or together) in the markup, described more below.

Information content might also use VoiceXML (www.voicexml.org) which is an XML based language for specifying voice dialogs, including audio prompts and text-to-

speech (TTS) for output and touch-tone keys (DTMF) as well as automatic speech recognition (ASR) for input. VoiceXML technology enables consolidation of voice and web applications. For example, it can be used with voice-only devices to access a voice portal, or used to facilitate multi-modal (graphical and voice) dialogs to VoiceXML enabled client browsers).

Preferably, the system (100 in FIG. 1) via the user manager 110 has the ability to read and process VoiceXML markup as well as convert from one markup (from WML for instance) to VoiceXML format. In addition, templates and/or wireless markup can be used to specify which parts of a web page are to be audible (i.e. converted to VoiceXML) and which are to be rendered visually by the browser. The server can interact with a VoiceXML gateway (much in the same way as it does with the WML gateway) to facilitate the VoiceXML based services.

Location based services might also prove to be very popular in this industry as they are well suited to mobile applications. Preferably, the server browser 110 via the user agent 110 has the ability to interact with the network entity supplying the location information via a defined protocol. The current coordinates of the electronic device accessing the network are preferably sent in the request headers to the content (web) server and/or be accessible via session cookies so that the device can easily utilize this location information. This functionality enables useful applications such as a restaurant locator that lists restaurants within a few miles of the user's current location.

Referring back to FIG. 1, the server browser 110 contains a cookie handler 112. Cookies provide a means of personalizing the information content that is retrieved by the

user agent 110 on behalf of the user of the electronic device 104. Preferably, the cookie handler 112 supports session and persistent cookies. Session cookies are valid for the current user's session and persistent cookies can expire after a pre-determined time specified in the cookie or be permanent. An added benefit to server side cookie processing is that the user is provided access to his or her cookies from multiple electronic devices and the user's cookies are not lost when the user changes electronic devices.

Referring back to FIG. 2, the user agent 110 translates the requested data content, if necessary, into a recognizable markup language for further processing. The markup language may be in the format of XML, WML, HTML, or any other markup language or technology (e.g., video, audio, image) that incorporates the features used by the present embodiments.

The translated information is then organized into a logically structured format for further processing by the QDOM 116. The QDOM 116 efficiently constructs a nodal structure. The use of the QDOM 116 enables a standard structured interface to the retrieved content that can be utilized by all modules of the server browser 110. The QDOM 116 can effectively and efficiently store the information content in a standardized structure for use by the normalizer, more described below.

Referring back to FIG. 1, the server browser 110 has script executor 120 for assisting the QDOM 116 in interpreting embedded script code in the information content received from the information source 102. The script executor 120 is preferably capable of supporting the European Computer Manufactures Association standard (ECMAScript

revision 3), which is most prevalent in the industry, but may also be capable of handling other scripting languages, known in the art, such as JavaScript, Jscript (Microsoft's extending implementation of ECMAScript), Visual Basic Script (VBScript), or WMLScript. The script executor 120 enables programmatic access to the QDOM 116 representation of the document.

This extension to the QDOM 116 can allow executed script code to modify the resultant document that is sent to the client browser 112, thus enabling dynamic content generation via scripting. Script executor 120 can also allow programmatic access to the cookies for a particular user, giving the content author the ability to create, modify or retrieve cookies associated with a given resource via script code. To interact with the user, communications between the script executor 120 and the client browser 112 is done via script events that are part of the application protocol between the server browser 110 and the client browser 112.

Referring back to FIG. 1, the server browser 110 has a Cascading Style Sheet (CSS) processor for supporting cascading style sheets. Cascading Style Sheets is now an industry standard developed and promoted by W3C (www.w3.org/Style) and supported by many of todays desktop browsers such as Internet Explorer. It defines a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents that describes how the document should be presented on the screen. By attaching style sheets to structured documents on the Web (e.g. HTML), authors and readers can influence the presentation of documents without sacrificing device-independence or adding new HTML tags. The CCS processor can apply elements of style specified in the content

received from the information source 102. For example, the information source 102 may specify style elements including, but not limited to, color, background, size, font type, font size and alignment.

Preferably, the CSS processor is capable of supporting the W3C specifications for CSS levels 1 & 2. In addition, the CSS processor interacts with the script executor 120 in a manner that enables the information source to specify dynamically changeable elements of style via a scripting language. The QDOM resulting from CSS processing represents stylized content that can be interpreted and visually represented by the client browser 112.

Referring again to FIG. 2, data content that has been transformed into a DOM tree is then forwarded from the QDOM 116 to the normalizer 124. Preferably, the normalizer 124 sends the DOM tree first to a template normalizer. If the template normalizer is unsuccessful at normalization, the DOM tree is then forwarded to an automatic normalizer where the data is normalized and then forwarded to the serializer 128 to be sent to the electronic device 104 via the connectivity manager 132, further described below.

The serializer 128 utilizes the normalized tree as input and produces a media stream targeted for a specific electronic device 104. Applying a style sheet or formatting rules to the DOM tree outputs a document (e.g., an XML document) that will be streamed to the electronic device 104. Preferably, the formatting rules are electronic device 104 specific and take into account display size, font types, color etc. as well as the particular markup language(s) supported by the target electronic device 104.

The server browser 110 has connectivity manager 132 for interacting with the client browser 112. Connectivity manager 132 sends and receives information to the client browser 112 using an event format and protocol such as a proprietary format (e.g., OBML, described below) and XML event messages. According to the requirements of the electronic device 104, the events may be translated via the event translator 136 (FIG. 1) to use an externally defined format and protocol such as WML and WAP.

Referring back to FIG. 1, the event translator 136 preferably provides the server browser 110 with compatibility with any standard client browser 140 or end user application that employs a known protocol such as HTTP. Current examples of such standard client browsers 140 include both WAP and non-WAP based WML browsers, HTML browsers, XHTML browsers, as well as both iMode and non-iMode compact-HTML (cHTML) browsers.

FIG. 3 is a diagram further illustrating exemplary uses for the event translator of FIG. 1. The event translator 136 can be utilized on the server browser 110 side or the client browser 112 side, depending on which client browser 112 or standard client browser 140 is utilized. According to the server browser 110 side, the event translator 136 operates as the interface between a standard client browser 140 and the information content data stored in a DOM format at the server browser 110. Preferably, the DOM is a QDOM 116 that identifies each node in the document using a unique value. According to the client browser 112 side, the event translator 136 can exist on the client browser 112 and provide an interface between the third party viewer and the micro-gateway 144 144. One skilled in the art would appreciate that the event translator 136 is not limited to the

server browser 110 side or the client browser 112 side, but can also operate between and externally to the server browser 110 and client browser 112, if so desired.

Preferably, the event translator 136 translates requests for information content to known events that can be used to generate or modify a DOM tree, dynamically assigns unique device identifiers to identify the information source 102 of standard client browser 140 events, sends events and receives responses to and from the server browser 110, and manages sessions and transactions (including timeouts, authentication, error handling etc.)

Referring back to FIG. 1, the client browser 112 can be hosted on an electronic device 104 such as a PDA, handheld PC, mobile phone or any device with sufficient navigation and presentation capability. The client browser 112 provides the user interface for presentation or rendering of the retrieved information as well navigational capability.

Furthermore, portions of the client browser 112 may be used by a standard or commercially available client browser. In such examples, the client browser 112 can provide distributed browser functionality that is compatible with the standard or commercially available client browser.

Moreover, the electronic device 104 is preferably validated by user/appliance ID, which can be stored within a database on a server. This ID authenticates the electronic device 104 and validates that it is allowed to access specific data content found in a particular data source.

In an exemplary embodiment, the client browser 112 can access information content via the server browser 110. Additionally, the client browser 112 preferably allows the user to submit information content or form data back to the information source 102. The form data is a response to a query posed by the data content of a particular document from the information source 102. Events containing the data for each component in the form are forwarded to the server browser 110 by the electronic device 104 where they are formatted in accordance with the content of that document by the server browser 110. The result may be an error notification (e.g. network timeout, bad data, etc.) or display of a new document received from the server browser 112 as a result of server side 112 processing of the submitted form data.

Referring now to FIG. 4, the client browser 112 preferably includes a microbrowser 148, event controller 152, and DOM store 116 according to the industry standard Model View Controller (MVC) representation. The microbrowser 148 is one example of an end user "View" application and represents information such as graphical or textual display, or audio to the user. The event controller 152 processes events to and from the server browser 110. The DOM store 124 is utilized for caching the information content received over the communications network 114.

The event controller 152 and the DOM store 124 operate as a micro-gateway 144 between the server browser 110 and an application 168 for interaction with the end user. In the exemplary embodiment the application 168 presents a typical browser interface allowing display and navigation of content, form interaction and submission and so forth. It should be understood, however, that the micro-gateway 144 can also support multiple

different end user applications 172 on the electronic device depending on their availability and the nature and type of content data delivered to it.

Examples of such end user applications include, but are not limited to, email, instant messaging, media players and other such plug-ins. Further, multiple different kinds of browsers designed for particular markup types (HTML, cHTML, WML, etc.) and so forth can also be supported by the micro-gateway 144. The micro-gateway 144 can also be used as a front end to enable access to web services by end user applications. In this regard, the application 168 utilizes a common API to the micro-gateway as a transport mechanism to relay requests and responses to the web service application information source 102.

In one aspect of the exemplary embodiment, the micro-gateway 144 presents an external interface to other applications 168 and 172 that consists of a well defined interface to the DOM Store 124 component and an interface to the event controller 152 using the same event model that is described below for communications internal to the distributed browser. In another embodiment, the micro-gateway 144 can be combined with the end user application 168 and 172 and use a more tightly coupled internal interface.

Additionally, the micro-gateway 144 can use an event translator (not shown in FIG. 4) to provide an interface to third party or commercially available applications such as HTML or WML or cHTML browsers. For example, according to an aspect of the present embodiment, a micro-gateway 144 and an event translator to cHTML can be used to provide an interface between the server browser component and the third party

cHTML browser known as Pocket Internet Explorer on a Pocket PC device running the Microsoft Windows CE operating system.

The microbrowser 148, renders the information content transmitted to the client browser 112 by the server browser 110. In the exemplary embodiment, the rendering includes visual representations (both textual and graphical) of the markup elements, but can be extended to provide other representations (e.g., audio) according to the capabilities of the electronic device 104. The format of the representations can be fixed by particular microbrowser 148 implementations such as a WML or proprietary (e.g., OBML) browser, or can be modified according to an XSLT style defined in conjunction with the content markup.

Typically the microbrowser 148 is used to directly display markup based content received through the micro-gateway 172. In addition, the markup based content can be used to adaptively tailor the microbrowser 148 according to directives contained in the information content. Adaptations can include embedding application logic in the content presentation, modifications to the interface (menus, titles, etc.) and other configuration of the browser application or device.

It should be understood, however, that an additional property of the browser is the ability to download and install other applications or plug-ins as needed to support non-markup based content, including images, audio, video, and multipurpose internet mail extensions (MIME) or secure MIME (S/MIME) document formats such as plain text, Acrobat (e.g., "*.pdf" format), Microsoft Word and so forth. Content of these types is

can be viewed through the use of these other applications or plug-ins by the micro-gateway 144 or microbrowser 148, or both 144 and 148.

Preferably, the event controller 152 is an event handler. Events to and from the server browser 110 are formatted according to the particular electronic device 104 in use. In the exemplary embodiment, XML event messages and a proprietary protocol are interpreted by the event controller 152 to manage data and events to and from the server browser 110.

The DOM store 124 provides a secondary cache of the information content stored on the server browser 110. The secondary cache preferably resides on the electronic device 104 to reduce the need to pass data between the server and client browsers 110 and 112, respectfully. The information content that the user desires is transmitted to the client browser 112 from the server browser 110. In the exemplary embodiment, however, once the information has been transmitted, it is stored for reuse. The information content is retained on the electronic device 104 while it is valid and while there is sufficient space to store it. The information content is stored in a DOM structure according to the W3C definition of an XML document and can be accessed by the microbrowser 148 or other applications on the client browser 112.

When the DOM is examined or modified, the event controller 152 preferably delegates the event to the DOM facade 118. These events include click, blurchange, submit, expand, etc.

In the case of a click event, the DOM facade 118 preferably examines and executes the content and/or script associated with clicking on a given node. In the case of

an bulrchange event, the DOM facade 118 preferably modifies the DOM so it reflects that the user has provided data. In the case of a submit event, the DOM facade 118 preferably examines the existing user input to formulate an appropriate form submission request. In the case of an expand event, the DOM facade 118 preferably delegates to a serializer to server the contents of a given folder to the client browser.

In cases where a standard browser (cHTML, WML, XHTML, etc.) is used, there is preferably no event controller. The requests from the standard browser are interpreted and translated by the event translator into events that the DOM facade 118 understands.

An "active" page is a page from the information source that is of higher than normal interest to the user of the client browser 112. It is given an active status in the cache and retained on the electronic device 14 when it otherwise might have been deleted for space or timing reasons. Where a document has been made part of the "active" cache (either through a push from the server or explicitly requested by the user through a client application such as the micro-browser), the content and any related data for that document can be accessed when the electronic device 104 is out of coverage.

Preferably, the client browser 112 handles "active" page content submissions separately and can store form data for those submissions to allow repeated attempts to submit that data until either the submission is successful or the user explicitly instructs the client browser application to delete the form data. Delivery of form submission data for the "active" content pages is guaranteed by the storage of the data in the client application browser.

Further, the client browser 112 retains the context of "active" page submissions and allows the user to access a history of chained interactions with the information source as a series of submissions of data and responses to the submissions. Note that each response can itself require a further submission and in turn generate another response, thus forming the chain. The status of an active submission in progress can also be viewed in the client browser.

Traditional HTML applications rely on continued connectivity between the client application and the information source. Normally a new page is sent to the client application in response to a user action such as a click on a link or submission of form data. Many applications include a series of chained form submissions where data posted with one form is carried forward to the next page by the information source. This is often done merely for the convenience of the user and is not strictly necessary for the application itself.

One common example is the completion of an order form where purchaser details, delivery address, credit card information or other information may be entered on separate pages but none of the values are used by the information source until a final order confirmation form (e.g., a "checkout" form) is submitted. Information posted in the leading forms is often passed through to subsequent forms using hidden fields. The information source takes the values submitted in the leading form and places them in fields that are not visible to the user but are submitted with the new form. In these types of applications, if the connection is broken or unavailable, then the application is blocked.

However, in today's wireless networks it is not unusual for mobile device to roam in and out of coverage areas or to be completely out of coverage.

In one embodiment, applications with chained form submissions are supported even when the connection to the information source is broken by allowing data to be passed through to subsequent (cached) pages on the client. An exemplary process for supporting the functionality includes:

- Input fields whose values are to be passed through are marked at the information source with the attribute "wireless:carryforward" set to "true" and have a unique value set in the "name" attribute
- Links or submit buttons used to navigate to subsequent pages in the application are marked with the attribute "wireless:loadhref" set to the URL of the subsequent page to navigate to
- All pages for the application are pre-cached on the client device
- When the user clicks on a link or form submission control with the "wireless:loadhref" attribute in the application, three things happen: 1) name value pairs are collected for all input fields with the "wireless:carryforward" attribute; 2) the page associated with the URL from the "wireless:loadhref" attribute is loaded from cache instead of attempting to submit to the information source; 3) values from the set stored in the first step are assigned to any input fields on the new page that have the same name as in the previous page

- Finally, when a form submission control without the "wireless:loadhref" attribute is clicked, all the values are collected and a form submission to the information source is made. As described above, this data may be submitted immediately if the client device is in coverage or may be stored for repeated submission attempts until coverage is restored.

The server browser 110 contains a Push Server which has functionality and interfaces enabling an application or system administrator to push information to the client browser 112. The pushed information can take the form of simple text alerts, web (multimedia) content from the information source 102 or configuration updates to the client browser 112 including but not limited to enabling or disabling feature options, adding bookmarks, modifying network preferences and changing the client browser's 110 look and feel. In addition to content, the server browser 110 can facilitate a push of updates/upgrades to components of the client browser or an update of the entire client browser application 112.

Preferably, the interface (API) to the Push Server will conform to the WAP Push Protocol specified by the WAP Forum. Accordingly, the push initiation may also be via a Multimedia Messaging Service (MMS) Server/Proxy-Relay. As described in that WAP Forum specification [WAP-205-MMSArchOverview-20010425-a] MMS Server is used to provide storage services for the messages while the MMS Proxy-Relay component interacts with the client browser 112 and with other available messaging systems. The MMS Server and MMS Proxy-Relay components may or may not be combined. Preferably, the push interface will be extended to support the ability to push a part of the

content from the information source 102 which replaces in-line only that part of the content page as rendered in the client browser 112 which acts as the MMS Client.

The *follow-me* service capability is a unique mechanism which enables pushed messages (alerts or other types of content) to reach the destination end user regardless of where that user may be or the type of device in use (e.g., PDA, phone, desktop browser, desktop email, instant messaging client etc.) This approach has many benefits, two of which include: 1) increasing the probability that the message will reach its destination and be read by the end user within the least possible elapsed time period, and 2) avoiding the "inbox overload" problem associated with broadcast messaging to many addresses since the message will only be stored in an active state on a single terminal (the one which acknowledged message receipt/read).

Smart addressing technology can be used in implementing the follow-me service concept. Smart addressing technology allows for a single user address to be associated with many devices such that the originating user or application needs only to specify one target address even though that address may in reality resolve to multiple unique addresses. In determining which target device should be tried and when that device should be considered as "not responding" there are a number of acknowledgement level checks and customizable timeouts that come into play. FIG. 32 depicts an exemplary algorithm used for implementing the *follow-me* capability.

The distributed browser 108 preferably utilizes send and receive events to convey information between the server and client browser 110 and 112, respectively. Events between these components can be classed as an events message,

acknowledgment/negative acknowledgment (ACK/NAK) message, security handshake request/response, and decrypt ACK/NAK message.

An events message preferably carries information between the server and client browsers 110 and 112, respectively. An ACK/NAK message is used to confirm or deny receipt of an events message. A security handshake message is used to transport information used by encryption and decryption routines. A decrypt ACK/NAK is used to report success or failure in the security routines.

The first byte of the messages preferably contains an identifier that uniquely defines which type of message is contained in the data being sent. The 2-byte integral values are always "little endian" or bytes at lower addresses have lower significance.

FIG. 5 shows the structure of an exemplary events message that is transmitted between the server browser 110 and the microbrowser 148 (i.e., via the network interface 156 and event controller 152). The event message may be modified or intercepted by the event controller 152. The event message includes header identifier, header version number, and the unique message identifier. Additionally, the event message also includes the event type such as response, original, push, or stop. Moreover, event data is included at the end of the events message.

The event data area of the events message may be compressed and/or encrypted as specified in the header component. When converted to plain text, the data area is defined as having the following exemplary structure:

<XML Events Message> = <Event Protocol Version><Event Separator><Session ID><Event Separator><XML Events><EOM>
<Event Protocol Version> = Integer
<Session ID> = <Server Session ID> |

		<Server Session ID><integer separator><Client Session ID>
<Server Session ID>	=	<Device Type><integer separator><Page ID>
<Device Type>	=	Integer (uniquely identifying different devices)
<Page ID>	=	Integer
<Client Session ID>	=	Integer
<XML Events>	=	<Event> | <Event><Event Separator><XMLEvents>
<Event>	=	<Event ID><field separator><XML Node><field separator><Attributes>
<Event ID>	=	integer (see table below)
<XML Node>	=	Integer
<Attributes>	=	<value> | <value><value separator><Attributes>
<value>	=	ASCII text
<EOM>	=	\r\n (0x0D 0x0A)
<Event Separator>	=	|
<Field separator>	=	*
<Value separator>	=	^
<Integer separator>	=	,

All events contained within an events message belong to the same session or page of content.

When there is only a single client browser 112 on the electronic device 104, the session id information needs only to specify an identifier for the server browser 110 session or content page. When there are multiple client browsers on the electronic device 104 (e.g. multiple browser windows, browser + instant message client, or other combinations), the events message should identify which client browser 112 it is associated with.

Preferably, the page id is generated on the server browser 110. When the client browser 112 has no valid server session id to send, a zero will be sent. Zero is not used as a valid session id.

For any Uniform Resource Locator (URL) data that is contained in an events message, standard URL encoding as is known in the art, is used to ensure that the information content does not include any of the characters special to a proprietary packet format (i.e., "|", "^", and "*"). For any XML content contained in an events message, standard HTML encoding, as is known in the art, is used (e.g., where characters can be represented by "&#n;" where n is the ASCII code for that character).

The node value maps directly to a node in the DOM tree (e.g., output of the QDOM 116 in FIG. 2) and signifies the node affected by the event. For some events (e.g. load, error) the node is set to 0, indicating that there is no direct connection to a particular node. Other events (e.g. expand, onblurchange, submit, onclick) have a direct correlation to an element in the QDOM and are targeted to that element by the value set in the events message.

The following table lists the different events that can be contained within an events message:

Event	**Attribute list**	**Description**
Cleaned	(none)	Notification to clients that old data has been purged so that clients can check any cached page references
Error	Error message	Description of an error on server
Expand	(none)	Client request for content of a normalizer folder
Expand	Document	Content of a normalizer folder
Load	URL, summary option, table option, JavaScript option	Client request for a new content page, including options on whether to normalize, include tables and allow JavaScript processing

Event	Attribute list	Description
Load	URL, summary option, table option, JavaScript option, document	Content page response or push from server, including options set in original request
Notify	Message data	Non-error message to client(s)
Onblurchange	new value for input element	The user has changed the content of an input element and moved focus away from it
Onclick	(none)	Click on link element
Reload	(none)	User wishes to force a reload of the current content page in the device application. The server will replace any existing session with a new load of the page from the web.
Stop	(none)	User request to stop any message identified by the MessageID in the header.
Submit	(none)	User has completed a form and is submitting all onblurchange data to the server
Authenticate	Realm	The remote HTTP server issued a challenge string requiring the user to prove possession of a valid user id and password for the realm
Authenticate	Authentication tokens	<username>:<password>, encoded in Base64 to be submitted back to the origin server
Alert	Message	Server initiated message. The device displays the alert message followed by an OK button.
Alert	(none)	Device returns when the user presses OK.
Confirm	Question	Server initiated message. The device displays the question followed by an OK and Cancel button.
Confirm	confirmation status	Device returns the button pressed.
Prompt	Message, default	Server initiated message. The device displays the message followed by a text input field and OK, Clear and Cancel buttons. The default message is displayed as the initial input.
Prompt	Button pressed, return string	If the user clicks the cancel button, return string should be null. If the user clicks the OK button, Device returns the value currently displayed in the input field.

To conserve bandwidth over the communications network 114 between the connectivity manager 36 and the radio interface 156, an events message may contain

more than one event. In one exemplary embodiment, all onblurchange events are stored by the microbrowser 148 until a submit event is generated. At that point they are bundled together into a single events message for transmission to the server browser 110.

FIG. 6 shows the structure of an exemplary transmission ACK/NAK message that is transmitted between the radio interface (156 of FIG. 4) and connectivity (132 of FIG. 2). An ACK message is preferably used when the underlying communications network 114 does not provide message delivery confirmation to the network interface 156 and connectivity manager 132. A NAK message signifies that although an events message was received, it could not be processed (e.g. because of low memory conditions).

FIG. 7 shows the structure of an exemplary message used by the security subsystems of the connectivity manager 132 and the network interface 156. The security handshake request and security handshake response messages are used to exchange public keys between the connectivity manager 132 and network interface 156. The security ACK and NAK messages are used to signify successful decryption of events messages that have been encrypted using the shared public keys. Receipt of a security NAK message at any time or a need to encrypt the data in an events message when public keys have not been shared preferably initiate a security handshake request.

The server browser 110 includes a QDOM 116. The QDOM 116 utilizes a in-memory representation of an document tree as a single mutable object or creation of a DOM does not require creation of a language object for every node of the tree, whether that language happens to be Java or C++ or another object oriented language.

The mutability of a QDOM 116 is preferred, because transformations of the DOM tree will be applied, resulting in a new tree structure within the same QDOM Object. The architecture of a QDOM 116 allows these transformations to be performed in an efficient manner with regard to both speed and resources used.

A QDOM 116 consists of an aggregation of N re-usable buffers that contain arrays of raw bytes. As the QDOM grows, additional re-usable buffers are added, only as needed. Some of the re-usable buffers contain binary information describing the DOM tree structure, tree dependencies, and references to information content data. Other buffers contain the actual content data.

FIG. 8 shows one exemplary embodiment in which separate QDOM 116 arrays 180 are used for the values representing the following properties of each element node 184: element name tag 188, parent node 192, previous element sibling 196, next element sibling 200, first child element 204, and first attribute 208. Similarly, each attribute node can be described with the following properties including attribute name tag, attribute value tag, previous attribute sibling, and next attribute sibling. Separate arrays can be stored in the QDOM 116 for attribute data, or it can be overlaid in the same arrays used for element data.

In another exemplary embodiment, the same information can be stored in structures for each node and attribute. The QDOM 116 contains one or more arrays of each of these structures. The choice of the means of storage depends on the functionality of the programming language employed for the QDOM.

The actual string names of the tags and attributes of tree elements are replaced by a corresponding value equivalent. A dictionary of the strings and their corresponding value is preferably built up as necessary to deal with a particular set of XML tags. For performance reasons, pre-compiled dictionaries can be used for the well-known markup languages, such as HTML or WML.

FIG. 9 shows a re-usable content buffer 212 that the QDOM array 180 references when information content that is not well known (such as plain text data) is used. The QDOM array 180 preferably stores the start and end positions of the text in the content buffer 212. In another exemplary embodiment, the content buffer 212 stores zero terminated strings and the QDOM array 180 stores the start position.

Preferably, the interface to the QDOM 116 is value based because every node (for example, see 184 in FIG. 8) of the tree has a unique value associated with it. All operations on the nodes in the tree that the interface provides can be carried out using that value as a reference to the affected nodes. All comparisons between nodes are also value based, so expensive string comparisons can be avoided.

Since the underlying structure of the QDOM 116 groups of raw bytes, it can easily and efficiently be serialized to and from permanent storage, thus reducing the amount of random access memory (RAM) memory necessary to support multiple users and multiple documents per user.

In situations where resources are limited, such as on a PDA, the QDOM 116 structure in the form of DOM store, XMLDocument, and XMLelement (124, 162, and 164 in FIG. 4 respectfully) can also be used efficiently by the client browser 112. The

QDOM storage is divided into smaller groups that are held in a permanent storage area (e.g., FLASH). Only such groups as are needed for the current operations on the QDOM are retained in or moved to a fast memory area (e.g. RAM). This provides high performance through an efficient use of the data in RAM while still minimizing the actual resource load in use at any one time.

A QDOM 116 is implemented as a re-usable object, so rather than deleting it and having a "garbage collector" reclaim a space occupied by the document, the QDOM 116 can be easily and efficiently re-initialized and used to store some other XML Document.

A number of preliminary tests have been taken to determine the time saved using the QDOM 116 as compared to the node-based interface of the standard W3C DOM. The tests show more than 100x improvement over other W3C compliant models.

The normalizer organizes the DOM tree into tiers or folders under headings that contain related content. The result is a set of hierarchical DOM node collections. The characteristics of font, font size, font color, hue saturation comparison of background and foreground color and Cascading Style Sheet or XSLT properties are used to determine the weight of a text node. The weight is then used to determine whether it will be inserted into a normalized document tree as a parent or child. The parent nodes become folder titles and the child nodes become the folder contents. Thus, higher weight document objects are pushed to the top of the tree so the user can decide whether to "walk" down the branch or not.

The normalizer dynamically streamlines and folderized the content automatically or via predefined additional rules to achieve automatically an experience similar to

reading a newspaper. The normalizer including the template normalizer and meta-tags allow the content source to be redefined once for all networks and device types. The alternative technologies in the industry are large cycle time, re-development of the content, often specific to one or more of the following: each device ergonomics, or a particular client-only browser, or a particular network type.

The goal to normalizing is to adapt desktop focus web content to handheld browsers. This requires filtering unsupported content, dropping unneeded content, reordering and partitioning content to improve navigation and application flow for display on a limited device. Some of the functions to normalization are folderize/partition content, drop content not required on a handheld device, reorder content, provide prompts/names to input elements

The normalization process can utilize a weighted heuristic and pattern recognition to create the contextual relationship between nodes in the source tree. The output from the normalization process is a hierarchical content tree. Preferably, the normalized tree is not specific to a particular presentation language. Therefore it can be transcoded for display by any type of client browser.

Content collapsing rules in the automatic normalizer utilize the previous page loaded to determine if similar constructs exist in a page which can be collapsed into folders or selectable input elements on subsequent loads. This is performed by comparing the previous page loaded with the current page. The trees of the documents are compared to determine if similar fragments (list of links, table, image) exist. The similar fragments

of the tree are collapsed into folders or select input elements. The effect is to conserve display space on the device.

Electronic devices can have limited display characteristics such as display size, font types, color etc. Most web content is tailored for display on desktop browsers which not only suppose a large screen, but also support a rich set of fonts, colors, and formatting constructs such as tables and frames. The normalizer adapts existing information content for display on the electronic device 104. The normalizer 124 includes an automatic normalizer 80 and template normalizer 84.

Referring back to FIG. 2, the automatic normalizer 80 in the normalizer 124 maintains the context of the information content before taking the electronic device 104 specifics into account. The automatic normalization process does this by organizing the information content into folders. The result of this approach is that sets of nested folders are created which the user can "walk" the information content on the electronic device 104. The titles of the folders are sent to the electronic device 104 first and the user can determine if the contents of particular folders are of interest. This not only increases usability in terms of reduced content to scroll through for the user and time spent scrolling through the page, it also optimizes wireless bandwidth utilization because less data is sent to the electronic device 104.

It should be noted that while the exemplary embodiments of the normalizer processes concentrate on the normalization of HTML content, including that generated by scripting technologies such as JavaScript, these same processes can be applied to other markup content. It is well suited to any XML content.

The automatic normalization process traverses a DOM tree from the QDOM 116 40 and creates a new, normalized tree. Preferably, the original DOM is transformed rather than copied, so that it becomes the normalized tree itself. This provides both efficient performance and efficient memory utilization.

The normalization process begins with the root node of the document and traverses the tree along a depth first path to maintain context at all times.

If a node is the beginning of a table, a table pattern recognition process is preferably executed. The entire table is weighted and pattern recognition criteria are compared to determine if the table matches a defined pattern. The table recognition criteria define a profile for different data table types. Each cell in the criteria is defined to be either greater than, less than or equal to a root table weight, a "don't care" or defined to contain certain nodes such as anchors or images. The root table weight can be derived from any cell in the table such as the cell at position row 0, column 0 or can be derived outside the context of the table. These criteria define the pattern that is attempting to be matched. If a pattern is recognized, the table cells are formatted corresponding to that pattern. If a pattern cannot be recognized, the weighted node processing continues.

The major part of the weighted node process is the maintenance of a weighted node stack. The first element of that stack is always the "DOCUMENT" itself, having by default the highest possible weight. The normalization process takes the next node from the DOM tree. The node is first filtered to determine if it has an effect on weighting or presentation. If the node is not significant it is preferably dropped. Nodes such as the HTML tag in HTML are not significant since the tag has no effect on presentation. Next

it is determined whether the node is a weighting node or a content node. Weighting nodes are nodes that affect the display of rendered content such as a bold or heading format tag. Some weighted tags may have a negative weight that allows nesting of the tags and emulates a hierarchy of nodes weights such as nested list items. Content nodes are nodes such as text nodes, input nodes, and image nodes.

Preferably, a node is identified as having no significance based on a projection of its visual impact when rendered in the client browser component. One or more of the following conditions might be sufficient for making such an identification:

- The node is an <image> element and the name of the image is part of a preset list of images to drop
- The node is an <image> element and the name of the image is equivalent to an image name that has statistically been seen to have no or negligible visual impact (e.g. it is common to use an image called "1.gif" or "spacer.gif" for spacing within tables, but when the table is unrolled they have no impact on the rendering of the page).
- The node is an <image> element whose width and height size attributes are specified in the HTML content and both are less than 3 pixels in size.
- The node is an <image> element that references an image whose actual size is negligible (less than 3 pixels in width and height). This can determined if the image has been previously requested and held in a cache.
- The node is an <image> element that is not clickable.

- The node is an <image> element whose name or ALT text matches a rule that identifies it as an advertisement. Often the names or URLs of advertisement images inserted in page content contain the element "advert" or "ad" or similar (e.g. the site http://www.nytimes.com contains numerous advertisements whose URLs all include a subdirectory named "ads" or "adx"). Similarly, advertisements often carry ALT text that simply says "advert" or "advertisement".
- The node is a frame or other block element that points to an external domain known to host an advertising server
- The node is a block element (, <table>, etc.) that has no visible children
- The node is an element that can not be supported by the requesting client browser component (e.g. and <object> element used for embedding Flash animation)
- The node matches on a template or style sheet rule that identifies it should be dropped.

Preferably these rules can be applied automatically by the normalizer component of the server or client browser. That is, these rules might be transparent to a user of the device. Alternatively, the rules might be visible to the user who can choose to enable/disable one or more of the rules, modify one or more of the existing rules and/or and add new rules to suit the content that the user is interested in.

When a weighted node is encountered, the node weight is added to the accumulated weight. When a content node is encountered it is assigned the accumulated weight and becomes a weighted node. The weighted node finds its position on the weighted node stack by finding the lightest element on the stack with a weight greater than his (node's parent). Stack nodes from that point on are preferably deleted from the stack. The new weighted node becomes a child node for that parent. When the node goes out of scope (e.g. if a TABLE is ended), the normalization process checks the weighted node stack to remove all nodes that belonged to the expired scope of influence.

For example, if the node on the weighted node stack is part of a table and the table scope of influence has expired, then that node is removed from the weighted node stack. However, if that node belongs to more than one scope of influence (e.g. it is part of one table nested inside another table), all scopes of influences are checked against that node and it is removed preferably when they are all expired. When the inner table ends, node stays until it replaced by a heavier node or the outer table ends.

The template normalizer 84 is a tool for recognizing patterns within a document through a DOM tree and applying changes to those patterns as specified in the template syntax. Preferably the template normalizer 84 is part of a larger process of normalizing all documents. One step of the larger process is an attempt to match and apply templates where possible.

The template normalizer 84 preferably handles numerous variations of the same document as dictated by the application logic that produces it. For example, if a certain data table could have various numbers of columns and rows, or even not be present in the

document at all, the template normalizer 84 is preferably still capable of dealing with those variations.

The template normalizer 84 is capable of recognizing an document from a set of potential documents that could be produced by the application. For example, if the application produces two completely different documents depending on whether a user is logged on or not, a single template could still recognize the outcome and apply its changes to either one.

The template normalizer 84 is also capable of recognizing a change in the overall structure of the document to determine when its rules are not applicable any longer. In that case, it will not attempt to apply the changes, but will convey the document to an automatic normalizer 80.

The template normalizer 84 uses a regular expression pattern-matching machine as applicable to DOM trees. The syntax of a template is strict XML as described by the World Wide Web Consortium (www.w3c.org) and includes a number of proprietary regular expression tags and attributes that describe how the template normalizer 84 should match and modify original content.

Tags:

<xgrp>	equivalent to a bracket in Regular Expressions
<xany>	equivalent to a wild card in Regular expressions
<xalt>	equivalent to 'or' in Regular expressions
<xadd>	signifies the addition of contained markup
<xnone>	equivalent to "empty" in Regular Expressions

Attributes:

"xtimes"	the number of times to apply a recurring pattern
"xtitle"	specifies a new title for a matched pattern
"xid"	specifies a name of an XML node
"xparent"	re-parent an XML node to a specified element

"xdrop"	delete a node from the tree
"xformat"	produce a formatting attribute for an input field
"xaction"	apply a user-specified algorithm to the branch of a tree

The normalization process preferably scans a dictionary of templates and an initial comparison is made based on a URL specified for the template and the URL of a document to be processed. If the URLs match, then the template normalization process begins.

The template normalizer 84 deals with a document or XML document in two steps: first it uses the regular expression pattern-matching machine to impose itself over a document and attempts to match the template to the document content. The match is performed using the regular expression tags in the template to apply standard regular expression algorithms. If that process fails, the changes specified by the template are not applied, and the document is conveyed to the automatic normalizer 80.

If the match process succeeds, the template normalizer 84 will apply changes as specified by the attributes of the matched regular expression elements. Changes include dropping nodes, creation of new folders or tiers, re-parenting nodes, assigning new titles to nodes, etc.

During the application of the changes, the template normalizer 84 can use a complex state variable syntax to refer to the parts of the document. These variables can be set as pointers to nodes in the DOM or counters. Resolution of a variable expression starts with the inner most expression and is processed outward. This is similar to list processing in the LISP programming language. The inner resolutions are used to resolve subsequent resolutions.

Where a counter is used in a variable expression, the variable will be resolved using the current value of the counter.

Resolution Example:

If $z = a and $ya = "data" the expression $(y($z)) would be resolved to

$(y$(z)) = $ya = "data".

$z gets resolved to a and this creates the expression $ya which is then resolved to "data".

Counter example:

If xcounter="x" and the first iteration of an Xgrp is occurring, then the expression $a($x) will be resolved to

$a1.

If a variable is used in the context of an Xtitle attribute, the value for the variable is derived from the text value of the DOM node that is referenced by the variable.

If the variable is used in the context of an xparent, then the variable is resolved to a DOM node to which to move the referring node as a child.

The xid attribute is used to set the value of a variable to the DOM node.

Template normalization involves matching a template DOM tree with an input DOM tree and applying changes to the input DOM tree. The automatic normalization algorithms may be called during the apply step of template normalization where specified by the xaction attribute.

With integration of the automatic normalization algorithms, a template can be utilized to drop and reorder content while calling the automatic normalization algorithms to partition the content into a hierarchical set of folders.

FIGS. 11 and 12 illustrate the flow of logic in the exemplary embodiment for determining when to apply the automatic, markup assisted or template normalizer 84 algorithms. It should be noted, however, that the process is not restricted to the current flow, but can include variations such as the application of the automatic normalizer before the application of the template normalizer.

At step 220, a DOM tree is preferably obtained from the QDOM and input into the normalizer.

At step 224, the normalizer determines if a template exists for the DOM tree of the document. If the template does not exist, the DOM tree is forwarded to the automatic normalizer for processing per step 232. If a template does exist, the DOM tree is forwarded to the template normalizer for processing per step 228.

At step 228, the template normalizer determines if the existing template matches the DOM tree per step 236. If the template does not match per step 236, the DOM tree is forwarded to the automatic normalizer for processing per step 232. If the template does match the DOM tree per step 236, the DOM tree is forwarded to the template application process per step 240 (and expanded in FIG. 110).

At step 244, if the template apply process fails, then the DOM tree is forwarded to the automatic normalizer per step 232. If the template apply process is successful, then the DOM tree has been normalized into a normalized tree.

At step 248, the normalized tree is forwarded to the serializer for transmission to the electronic device via the connectivity manager.

At step 232, the automatic normalizer is applied.

At step 252, it is determined if the DOM tree has normalization markup or meta-tags. If the document does not have normalization markup or meta-tags, the DOM tree is forwarded to step 260. If the document does have normalization markup or meta-tags, the DOM tree is forwarded to the markup assisted processing per step 256.

At step 256, the normalization markup or meta-tags are utilized to assist in normalizing the DOM tree into a normalized tree. The normalization markup can provide instruction to the automatic normalizer to create the normalized tree. The normalized tree is forwarded on to the serializer per step 248.

At step 260, the DOM tree is normalized using a "best guess" processing. After step 260, the DOM tree is normalized into a normalized tree and forward on to the serializer per step 248. According to FIG. 11, the template apply process per step 240 is further illustrated.

At step 264, the template apply process determines to use the automatic normalizer process per step 268 or the template rules process per step 272. If the template apply process determines to use the automatic normalizer rules, the DOM tree is forwarded to step 268. If the template apply process determines to use the template rules, the DOM tree is forwarded to the apply the template rules per step 272. If the template rules are applied per step 272, the DOM tree has been successfully normalized and returns to the normalization process per step 288.

At step 268, the normalization rules such as user profile, statistics, keywords, history, are applied and the DOM tree is forwarded to step 276 to determine if the document contains normalization markup. If the document does not contain normalization markup, the DOM tree is forwarded to markup assisted processing per step 280. If the document contains normalization markup, the DOM tree is forwarded to the "best guessing" processing for applying pattern recognition and weighting heuristics to the DOM tree and forwarded to the normalization process per step 288.

Some devices (e.g. those with small screens) do not support the display of tables as in a traditional desktop browser. The problem is determining how to extract content from the table into a linear form so that it is presentable on the device. Table pattern recognition looks for tables that conform to some of the more common uses of the tables for data presentation. A weighting and comparison of table cell nodes in the DOM tree against each other is used to determine the order in which to extract cell data from the DOM tree. The comparison of weights attempts to determine if the nodes match a recognized table pattern. Cell weights are assigned in the same manner as the general normalization process, but other characteristics are also compared. 'Other characteristics' preferably includes the type of cell element (table heading or table data) and if the table cell contains an anchor or an image, but may extend to any particular attribute of the cell data.

FIG 12 shows an example Table Pattern as formatted in the original content for a wide screen display (desktop) and reformatted for a small screen device. In this table the first row of table cells contain heavy weight text nodes. These nodes will become

headings/parents for the columns. The following rows will become children of the parent nodes as shown in the small screen display result.

Images can be used with a number of different elements in the original content such as anchors, buttons, images and image maps. If the element containing the image is contained in a table cell after the normalization process, its maximum allowable width is the width of the cell. If it is not contained in a table cell, its maximum allowable width is the available screen width. If an image is wider than its allowable maximum width, it will be scaled down to the maximum. If an image is not wider than its maximum width, it is not scaled.

The general normalization rules may not apply to all web sites. Therefore it may be possible to specify specific rules on a URL basis. Preferably, URLs can be matched exactly or partially, such as in the matching of the domain name component only.

Preferably, the templates and XSLT style sheets used by the normalizer can be created and registered for use in a number of ways. In one embodiment, they are created using an external application that works like an HTML editor such as those generally available in the industry (MacroMedia's Dreamweaver, Microsoft's FrontPage, etc.). The original HTML content can be rendered either directly in the editor application or through an external simulator as it would appear in the client browser, and the user can choose from a menu of options to apply the functions of the template normalizer (matching content, dropping content, moving content, unrolling tables, etc.). Additionally, the same feature set may be implemented as an add-in to a commercially available HTML editor such as MacroMedia's Dreamweaver, Microsoft's FrontPage, etc.

In either case, external file might be generated that includes the template or XSLT information. This file may then be registered with the server by an administrator and can be used by the normalizer in the process described above and, for example, in FIGs 10 and 11. Registration might involve associating the template or XSLT file with the URL of the page and recording that association in a database on the server. Preferably the administrator of the server browser can access the list of templates to modify, delete and enable/disable them.

Beyond the process described above which allows specification of information in advance that applies to any user utilizing the server browser component, an individual user might specify and register template information for use by the normalizer. In one aspect of this process, any user can use the same methods and applications as described above, submitting and registering the template with the server for either public use or their own private application. In another aspect, the user can utilize the client browser to create a template for any page that they have requested. Preferably, the client browser supports a "customize" mode that can be selected after any page has been loaded. In that mode, the user has access to the functions that can be applied by the normalizer (matching content, dropping content, moving content, unrolling tables, etc.) and can see the effects those functions as they are applied. Preferably, the user can select and highlight one or more elements of the document and choose from a popup menu of actions that the normalizer can perform. When the user completes the "customize mode" either explicitly or through a request of another page, the user is given the option of saving the customizations in a template. Preferably the template is saved and can then be

registered either on the client device or sent to the server where it can be applied privately to the user who created it or generally for all users. Preferably, information is kept on the client that allows the user to review, modify and enable/disable the templates that they have created.

It should be apparent to anyone skilled in the art that while there are numerous performance benefits in having the templates registered on the server because content can be reduced or modified to limit over the air transmissions requirements, there are also similar benefits to applying the same templates on a client browser when connection to an information source both through a server or directly in proxyless mode. In one aspect, although applying a template on the client to drop HTML content must still request that part of the content as part of the whole document, the content that is dropped may contain directives that would have otherwise made secondary requests for more content, such as JavaScript, CSS or images.

In oneembodiment, the normalizer component including the management and application of templates is part of the server browser component and resides on a separate machine to the client browser component. This component is separable, however, and might alternatively reside on a PDA or other client device with the client browser as described above. Further, the normalizer component may be extracted to a third entity that a user may install as a personal server on their desktop or other separate computer. This personal server may include all of the server browser components as shown in FIGs. 1, 2 and 3 or, as described here, it may include only part of them and work in conjunction

with a public server containing the other components. The personal server might store a registration of a specific configuration for a particular user, which might allow for:

- keeping the configuration information private to the user
- allowing management of a large number of templates that may not fit on a resource constrained PDA
- locating the personal server on the same side of the wireless network as the origin servers, it allows for the full benefit of network optimization

The concept behind adaptive browsing is that most users hit certain links or objects in a document more than others. The user's statistics can be used to increase the priority/weight of an object in the tree in order that the user be displayed that object first. This allows the page to be adapted to the user's tendency/habits. This mechanism will aide in browsing efficiency for the user. Content nodes will accumulate more weight based on the statistics on, for example, the click count on an anchor.

A User can provide keywords for his profile that can be used to assign weights to the document content. Content matching any keywords would be pushed to the top of the tree by assigning higher weights to the matched elements.

Further, the statistics collected on historical user actions can also be used to minimize over the air traffic by predicting a common navigational path for the user and pre-empting that on the server before sending any content to the client. This is of particular advantage on sites where the top level page appears consistent to the user, but the underlying targets for links change. This often happens, for example, on shopping

sites where the URL for the second level page can be dependent on date or session information and changes accordingly.

In such cases, it is not possible to reliably bookmark the second level of navigation even though the action to reach it is always the same. In this example, the server browser can detect that the same user always follows the load of the top level page with a click on a link named "current specials" (for example) and can save the user the time and possible cost of loading and displaying the top level page. The server browser loads the top page and if it finds a statistical record of a click on the same link in the past, the user click event is simulated to request the second level page. That page is then sent to the client instead of the top level one. The server side navigation may extend beyond a single level where the statistics show the user predictably clicking to advance through multiple pages on a site. FIG. 31 shows an exemplary process for determining when to automatically navigate on the server. Performing the navigation on the server might therefore save both time and network transfer costs for the user.

In another aspect, statistical and historical data can be used to pre-fetch content by the server and/or client browsers in anticipation of the user. For example, an enterprise server might access a field service application always requests a list of current tasks at or near 8am every weekday. Based on a historical review of the requests, the server browser may detect the recurring request and pre-load the data at exactly 8am. Further, the server can proactively push the pre-loaded page to the client device in advance of any actual request by the user. In this case, when the user makes a request for the page at, say, 8:10 am, the content will already be in the client cache and can be rendered

immediately. Over and above simply forcing the server to push the content to all users at the same time, this allows the users to "train" the server to match their use of the application and can compensate for a variety of uses of the application in different work environments, time zones, work shifts, etc.

Further, the statistical data collected on navigation can include the extent to which a user scrolls pages after loading them and before acting on the page (clicking on a link, entering form data, etc.). Many sites require a user to log in (e.g. hotmail, eBay, Amazon) and the first action of the user is to scroll to the login fields to enter the data. The client browser component can detect that the same level of scrolling always occurs on such pages and record a marker for that URL so that the navigation can be automatically played back on subsequent requests for that page. Preferably the client browser component also allows the user to manually create such markers to be associated and applied to particular URL responses. The marker can be calculated as a scroll position in pixels or associated with a particular node/element in the content itself.

Markup assisted normalization defines a wireless specific markup that conforms to HTML 3.2 and 4.0 specifications and does not affect the page for display by other HTML browsers.

The markup defines a set of attributes that can be introduced to existing HTML content. These attributes are triggers for the automatic normalization process to perform certain operations such as move content or create a folder.

TAGS:

HTML <meta> tag - <meta wireless:ml="true|false">

If a meta tag exists in the <head> of the HTML page with the wireless:ml attribute set to true, the automatic normalizer will be triggered to perform markup based normalization.

HTML tag

A div tag with wireless attributes is used to encompass blocks of HTML which will be processed by the automatic normalizer. The wireless attributes assigned to the div tag will not affect display of the content on HTML desktop browsers.

The markup is extensible in that other attributes may be added in the future to trigger other automatic normalization behavior such as wireless specific javascript.

HTML <style> tag

The style tag is used to encompass arbitrary blocks of HTML that will only be processed by the automatic normalizer. Preferably, the content encompassed by the style tag will not be displayed on HTML desktop browsers.

Attributes:

 attributes

wireless:action: (default behaviour = autonormalize)

	drop the content encompassed by the div from the wireless content
colbycoltablenormalizer	call a formatter on a table which extracts the data column by column
rowbyrowtablenormalizer	call a formatter on a table which extracts the data row by row
Folder	create a folder with a given title which will contain the encompassed content
Mark	mark a place in the content with an id
Move	move the encompassed content to marked position or folder
Include	include a block of html defined in a style block

McDonnell Boehnen Hulbert & Berghoff
300 S. Wacker Drive, Floor 3200
Chicago, IL 60606
Telephone: 312-913-0001

Autonormalize	call the automatic normalizer on the encompassed content
Passthru	pass the content encompassed by the div to the handheld as is

wireless:id:

sets an identification string for the folder, mark, move or include action.

wireless:title:

used to name a folder.

wireless:href:

used in conjunction with the menu action and the folder action to define an anchor/href link in the wireless page

<style> attributes

wireless:id:

an identification string for the block of html which can be included into the wireless page.

For example:

```
<html>
    <head>
    <meta wireless:ml="true"/>
      <style wireless:id="html block">
       block of html which can be included in the wireless page
      </style>
    </head>
    <body>
      <div wireless:action="include" wireless:id="html block">
     </div>
    </body>
</html>
```

Referring back to FIG. 1, the distributed browser 108 preferably appears to the information source 102 as a client interface, such as a PC that is browsing an Intranet or Internet. However, when the server browser 110 receives the information content plus script code, it can create an XML based Document Object Model ("DOM") tree structure. This structure can organize the information content into a structured format that is interpreted and processed by the normalizer to organize the information content into folders or tiers. After the information content is in a structured format, the server

browser 110 can forward the structured format to the client browser 112 hosted on the target electronic device 104.

Furthermore, information content can be transmitted from the server browser 110 to the client browser 112 without first receiving a request from the client browser 112. Therefore, information content such as in channels of desirable content like sports, weather, and stocks can be forwarded to the server browser 110 at periodic intervals. Additionally, the information content sent to the client browser 112 can include a note. Such a note could inform the user of the client browser 112 that additional information content is stored at the information source 102 or server browser 110, or both.

In any markup language (e.g. WML, HTML, cHTML, XHTML, etc.) the source content contains two distinct types of information – information that is visible to the user and information that carries processing instructions to the browser or viewer (e.g. a URI for new content to load when an element is selected, rules governing how to submit form data to the content source, etc.). The latter type of information, while not visible to the user, often occupies more than 50% of the total source HTML code.

Through the use of Object Based Markup Language (OBML), a proprietary markup language, an object aware distributed browser can take advantage of this distinction of information to leave the majority of the processing information on the server side and only send the end user visible information to the client browser. Under this scheme, the client version of the DOM will thus represent a functionally equivalent, but stripped-down version of the original DOM, allowing for efficiency in both content transmission to the device and event handling between the server and client components.

In order to support this scheme, every node in the original server side DOM is assigned a unique number. Every DOM node that has a processing instruction associated with it (e.g. in HTML this could be a link to follow when it is clicked), a "P pointer" attribute for that node is sent down to the client browser. Since it is unique, the P pointer can identify the element back to the server.

For example:

A DOM on the server side contains the following anchor information at node 1234

```
<a href="/hockey/nhl/2001/playoffs/news/2001/04/15/kings_red.wings_ap/?s=4"
>Kings beat Wings 2-1, end 14-game playoff skid</a>
```

When formatted in OBML, this node is represented as

```
<a p="1245">Kings beat Wings 2-1, end 14-game playoff skid</a>
```

This represents a significant savings on the amount of data that needs to be sent to the client over a network (e.g., wireless network). An event back to the server indicating that the anchor has been clicked on/selected need only carry the value of the P pointer. Note that a P pointer can also be used to identify the elements for other control events such as an input field or form submission button.

Similarly, OBML provides support for the folders that are the output of the normalizer through the use of an "x" pointer attribute. The presence of this X pointer indicates to an object aware browser that an element is an object representing a group of child elements and that those children can be requested from the server.

For example:

If the anchor shown above was identified as a folder title by the normalizer, it would be represented in OBML as

Kings beat Wings 2-1, end 14-game playoff skid

As with the use of the P pointer, events between the browser components requesting or delivering the content for expansion of the folder identify the element using the X pointer value only.

Note that an element can have a P pointer alone, an X pointer alone or both attributes together.

In the exemplary embodiment, the event translator can provide an interface to a WAP gateway and translates events in the following manner, an HTTP GET request for a URL of the format *p* + *digits* where digits is a number. (e.g. p174) is translated to an "onclick" event targeted at DOM node 174, an HTTP GET request for a URL of the format *x* + *digits* where digits is a number. (e.g. x174) is translated to an "expand" event targeted at DOM node 174, an HTTP POST request for a URL of the format *s* + *digits* where digits is a number. (e.g. s174) is translated to a "submit" event targeted at DOM node 174, an HTTP POST request containing postdata name value pairs in the format *n* + *digits* = somevalue (e.g. n323 = foo) is translated to an "onblurchange" event targeted at DOM node 323 with value "foo".

An HTTP POST request containing postdata name value pairs in the format somename = *n* + *digits* (e.g. myName = n323) is translated to an "onblurchange" event targeted at DOM node 323 with value "1".

In another embodiment, XML documents or other documents may also originate from the electronic device 104. For example, a user may scan a product bar code with a electronic device 104 to produce a document with a client application on the electronic device 104, and send the document via the gateway cluster 18 to an e-commerce transaction server for processing.

FIG. 12 further illustrates the process of the distributed browser 108 of FIG. 1 by showing a screen shot 300 of an exemplary web page (www.yahoo.com) on both a standard PC-based desktop browser (e.g., Microsoft's Internet Explorer) and a screen shot 304 on the RIM 857 electronic device. To view the yahoo.com web page on a PC-based desktop browser, a user might use a mouse and/or keyboard to enter the Universal Resource Locator (URL) that identifies the page.

The same user can enter the URL in the RIM device to retrieve the same information as shown on the screen shot 300. However, the screen size of the RIM device is smaller than that of a typical computer screen of a desktop computer. To accommodate the RIM screen size, the information is displayed such that the important information is displayed first. In this example, Personal email, Departments, Stores, Features, Arts & Humanities, Business & Economy are displayed first. The user can then appropriately navigate through the rest of the Web site using events such as by clicking, loading, changing, etc. In this example, the Business & Economy folder was expanded to the more information relating to the Business & Economy section, described more below.

Upon receipt of the event from the RIM device, the server browser proceeds to perform the request. The user agent retrieves the information corresponding with the

URL and appropriately forwards the bit stream to the QDOM. The QDOM then utilizes the QDOM to transform the data content into a DOM tree and forwards this tree to the normalizer. The normalizer (automatic, markup assisted and/or template) creates folders of content, typically by utilizing the characteristics of font, font size, and font color to determine the weight of a text node. In general, the titles of the folders are sent to the electronic device 304 first. In some instances, however, folder content can be sent down to the electronic device with the folder titles. This can occur based on a request by the user or through a determination by the serializer that the content is sufficiently small to be sent in the initial load.

In this example, the automatic normalizer identifies elements that are of higher importance or visibility on the desktop display and uses them as folder titles. Elements identified by the normalizer are bold font items (e.g. "My Yahoo!" and "Auctions"), table headings (e.g. "Departments") and larger font items (e.g. "Business & Economy"). A folder title can be derived from both ordinary text (e.g. "Departments") or hyperlinks (e.g. "Business & Economy"). In the latter case, the link property is preserved on the device.

For example, assume the user has selected "Business & Economy" and requested an expansion of the folder. The electronic device 304 may already contain the content for this folder cached in the micro-gateway (FIG. 4), in which case it can display it immediately as shown. If the content is not cached on the electronic device 304, however, a request is sent to the session manager where the entire DOM is cached. The content of the folder being expanded is extracted by the serializer and sent to the electronic device where it can then be displayed. It should be noted that the normalizer is

not restricted to a single level of folders but will produce an appropriate level of nested folders depending on the nature of the original content.

In accordance with the present embodiments, the server browser interacts with the network content and performs the primary functions unique to this architecture such as automatic translations, object based browsing, and object based personalization. In addition, the architecture uses XML as the protocol to exchange information between the various systems. Furthermore, the media managers can support instant messaging capabilities between desktop browsers and multiple electronic devices. Thus, the data is appropriately sent to the electronic device. Furthermore, the user is allowed to navigate through the Web site.

The normalizer also reduces the data content sent to the device by allowing the user to browse only the content of interest. It can achieve this by partitioning the web page content into expandable objects. The partitioning of content is achieved by a normalization process in which normalization rules are applied to the markup language tags, such as HTML tags. This feature enables the user to essentially walk a document tree and only query for content of interest.

Some tables which are small enough for an appliance may be sent directly to the appliance. These simple tables can be displayed as a table directly on the screen of the appliance. The default normalization for a table extracts the content of the table into an expandable object-based structure. Simple data type tables should be displayed on the device using a table control. Tables used for displaying for formatting or complex data (nested information, multiple fonts, mixed information such as text combined with links

in the one cell, etc.) are not required. Limited support for merged cells in heading rows is required.

In addition, specific rules for the optimizing the presentation of content designed for a large (desktop) screen on a small (pda or pager) screen may be implemented. Content that has been designed for a large screen is commonly formatted using tables and/or frames so that it takes full advantage of the width of the display. These can also be nested for further effect. The invention reformats and presents these layout constructs on the small screen in such a way that the original meaning and context of the content is preserved.

For target devices that support horizontal scrolling and complex display constructs such as tables, the goal is to selectively remove the need for horizontal scrolling where it is not required to preserve the context. Where the context requires that the content be wider than the screen (e.g. a wide data table), horizontal scrolling is kept to a minimum. For target devices that do not support horizontal scrolling in a user friendly manner (e.g. where the primary input device is a trackwheel commonly used for vertical scrolling), tables may be removed entirely.

Each sub-component of the original area (frame, table row, table cell or nested table) is considered separately and may be preserved, resized or replaced. In one embodiment, the components are considered in the order that they are defined in the original content, so that, for example, table components are always treated in a left to right, top to bottom order.

If a table contains nested tables, it is "flattened", preserving only the most nested tables for target devices that can support them. A cell that is flattened in this manner is replaced by the content within the cell and separated from the content of the next cell by a space. Each flattened row is separated from the next row using a break (
) element. If a table contains no nested tables and the target device supports table constructs, then the entire table construct is preserved.

If a table is preserved and is wider than the screen on the target device, then the content is checked to see if adjustments can be made to fit it to the screen width. This check is based on the contents of each cell, the number of columns in the table and the width of the target screen. Each cell is assigned an initial width determined by dividing the table's width (as specified in the original content) by the number of columns in the table. If the width of the table is not specified in the original content, then each column is assigned an initial width equal to half the target device screen width. If necessary, a final step is taken to reduce the column widths so that the table is not more than 4 times the width of the target screen. For tables with a single column, the width is set to the screen width.

Using the initial value as a guide, the width of every cell is then adjusted to fit the actual content of the cell, including shrinking empty cells as much as possible. Finally, the width of each column is set as the widest cell in the column (taking into account spanned columns). That width is then assigned to each cell in the column.

Similarly, frames are considered in the order that they are defined in the original content, so that they are treated in a left to right, top to bottom order. Each frame is

replaced by the content within the frame and separated from the content of the next frame using a break (
) element.FIG. 13 shows a flow chart of an exemplary embodiment implementing processing for selective horizontal scrolling. The input DOM 300 is analyzed to determine whether it contains frames at 302. If frames are used, the frames are flattened 304. Preferably, each frame is replaced by the content within the frame and separated from the content of the next frame using a break (
).

Once the existence of frames are determined, the existence of tables in the content are analyzed. If tables are utilized 308, the device is checked 310 to determine whether the devices supports table markup. If the device does not support table markup, the table(s) is flattened accordingly 312. If the device supports table markup, most of the nested tables are kept and the outer tables are flattened 314.

FIG. 14 is a flowchart of an exemplary process for nesting that can be used in conjunction with the flowchart of FIG. 13 to implement selective horizontal scrolling. FIG. 14 shows further shows the table processing of 314. The input table 320 is first checked to see if the end of table has been reached 322. If the end of the table is reached, the table processing is complete 324. If the processing is not complete, the next row 326 and cell from that row is analyzed 328. The cell is analyzed to determine if the cell contains a nested table 330. If the cell contains no nested tables, the cell contents are output 332 and the row checked to determine if the end of row is reached 334. If it is not the end of the row, output space 336 and the next cell is fetched 328 to repeat the process. If the cell contains a nested table, the nested table is fetched 338 and analyzed to

determine if the table contains nested tables 340. If the table contains nested tables, the table is flattened 342.

FIG. 15 is a flowchart of an exemplary process for table size calculations that can be used in conjunction with the flowchart of FIG. 13 to implement selective horizontal scrolling. Referring to Figure 15, the input table 340 is first checked to determine if the table is wider than the available screen 342. If the table fits within the screen, the table is ready to be drawn 344. See FIG. 13.If the table is wider than the available screen, additional processing is performed to selectively reduce the horizontal scrolling. The table width is checked to determine if a table width has been specified 346. If the table width is specified 346, the initial column width is calculated 348.

According to a preferred embodiment, the initial column width is calculated 348 according to the specified table width divided by the number of columns. If the table width is not specified, the initial column width is set to half the screen width according to a preferred embodiment 350. After the initial column width is calculated 348, 350, the table width is compared to 4 times the screen width 352. If the table width is greater than 4 times the screen width, the column is scaled so the table width is equal to 4 times the screen width.

The table is then checked 356 to determine if the end of the table has been reached. If the end of the table has been reached, the column width is set to the widest cell in the column 358. The cell width is set to width of the columns containing cells 360 and the process is then ready to draw 344. If the end of the table is not reached, the next row is fetched 362 and then the next cell 364. The cell width is set to the content spacing

divided by the column spacing plus the cell padding 366. The end of the row is still checked 368. If the end of row has not been reached, then the next cell is processed 364, 368. If the end of the row is reached, the end of table is checked 356 and either the next row is fetched 362 or the process is ready to draw 344.

FIG. 16 shows an example of original content for a typical screen display (desktop) and how selective horizontal scrolling reformats content for a small screen device. In the first table of FIG. 16, the nested table is simply fitted to the screen width for the small device. The outermost table content is fitted to the screen width by flattening the cells. This is achieved by replicating the content of each cell in the table and separating them with spaces between the cells within a row and breaks between rows. When the cell content is rendered to the screen, text is wrapped at the screen boundary. The end result is that horizontal scrolling is not needed. At the bottom of the page, the wider nested table goes beyond the screen width. The nested table is not fitted to the screen width. The horizontal scroll bar shows and can be used to view the outer portions of the content.

FIG. 17 shows selective horizontal scrolling exemplary handling of frames. As shown, frames are laid out left to right, top to bottom.

The left to right, top to bottom layout of content in frames or tables can result in the display of a significant amount of content before the "main" part of the page. FIG 21 shows some examples of where this can occur. As can be seen, it is common to surround the main area of the page with sections that are "navigation bars" for access to other areas in the domain or other sites. These bars may be either horizontal across the top of the

page or vertical at the left or right of the main area. While these sections include content that needs to be kept to retain the full functionality of the site, it is often the main area that is of initial interest to the user. In order to speed the initial use of the page, the user would like to quickly skip over these navigation areas.

In one exemplary embodiment, the areas are rendered fully, preferably using selective horizontal scrolling, but the initial view of the page is automatically scrolled to bring the main area into view. In another exemplary embodiment shown in FIG 21, the area occupied by the navigation bars is minimized so that the main content area shows on the first screen. Preferably, sequences of links are rolled up into a single select list so that the full sequence can be seen be popping up the select list, but the set only takes one line of the display area. Other content in the display area that separates the link sequences, such as isolated images, are shown in order.

The recognition of navigation bars, as described above, may be based on the identification of one or more of the following rules in the underlying HTML:

- Frame at top of screen is less than 150 pixels high (top navigation bar)
- Table row(s) at top of screen are full width of screen and less than 150 pixels high (top navigation bar)
- Frame at left/right of screen is less than 150 pixels wide (left/right navigation bar)
- Table cell, cells or column at left/right of screen is less than 150 pixels wide (left/right navigation bar)

- Each table cell at left/right of screen contains only a single HTML element (left/right navigation bar)
- Table cell or cells at left/right of screen contain content consisting of a set of links or images separated by
 elements (left/right navigation bar)
- Table column at left/right of screen contains content consisting of a set of table cells each of which has a single link or image element (left/right navigation bar)
- Frame at left/right of screen contains content consisting of a set of links or images separated by
 elements (left/right navigation bar)
- Table row or frame at top of screen contains content consisting of a set of links or images separated by whitespace characters or a recurring separator character (',', '|', etc.)

These rules are provided merely by way of example, and other rules might also be used to recognize navigation bars.

In another embodiment, the user of the client may be choose to view frames laid out in a manner similar to current desktop browsers (e.g., Internet Explorer, Netscape). In order to render the content in a manner suitable to small or low resolution screens, the size of the frames may be adapted to suit the content rather than the size specified in the original HTML for the page. Frame sizes are designed for larger, desktop size screens and can be specified as having an absolute size in pixels (FIG 22) or a percentage of the screen width (FIG 23). FIGs. 22 and 23 show the effects of following these specifications with a small screen and the poor results. In FIG. 22 the page's left frame

420 has a fixed width, leaving the rest of the page 421 for the main content area. It can be seen that when simply honoring the original HTML on the small screen it is only possible to see part of the main content area 422. Similarly, in FIG. 23, the page's left frames 430 have a relative width specification.

In this case, honoring the HTML specifications on a small screen results in the left frames 432 being where most of the content is hidden and consequently difficult and inconvenient to view. In both cases theframe scrollbars 423, 433 would occupy a significant portion of a small screen. FIGs 22 and 23 also show an exemplary embodiment 424, 434 that solves these problems. Preferably, the frames are sized so that all the content contained within each frame fits inside that frame. The result in both cases is that only page level scrolling 425, 435 (as opposed to frame level) is required.

FIGs. 24, 25, 26 and 27 are flowcharts of an exemplary process for calculating frame sizes so that scrolling within frames is not required.

In another exemplary embodiment, the frame level scrolling is preserved and the sizes (relative or proportional) as specified in the original HTML are honored, but the content is made accessible by mapping to a virtual screen size equivalent to common desktop display sizes (say, 800 x 600) rather than actual screen size. In this case, both page level and frame level scrolling may be required to view all of the content.

FIG. 18 shows the handling of images in selective horizontal scrolling mode. The larger images are scaled to the screen width. The smaller images are not scaled.

The capabilities of small screen devices continue to improve in both the areas of hardware (e.g. screen resolution) and software (e.g. the ability to switch between portrait

and landscape modes). As these features become available on resource rich devices, it becomes possible to use these capabilities to display some pages in a usable "desktop layout". The content is not only rendered in the same way as a desktop browser (e.g., Microsoft Internet Explorer) would be rendered, but enough of it is visible and easily accessible that an unfamiliar user can easily navigate within the page to find the content of interest to them.

Preferably, the characteristics and current mode of the device that it is running on can be examined to automatically determine which is the best display mode for a particular page. FIG. 28 is a flowchart of a process for using the initial visibility of form input controls to determine the best view for the Google search page. It can be seen that while the desktop layout works in landscape mode, it does not produce as good a result as the selective horizontal scrolling layout when the device is in portrait mode. Other factors that may be used to automatically determine the best view mode include:

- the amount of content on the page
- the overall size of the rendered content on the page
- the size and arrangement of block elements such as tables, forms or div
- the use of CSS style information (e.g. absolute positioning)
- the presence or lack of DHTML (dynamic HTML) in the content
- particular markup or metatags within the content that direct the browser to prefer one mode to the other
- the use of the client in a proxyless mode vs. using a proxy server that is known to adapt content

These factors a provided merely by way of example, and other factors might also be used.

FIG. 19 shows an exemplary handling of handling of popup windows. The original page is a single window. The popup creates tab overlaid on top of the original window. Clicking on the popup transfers data back to the original page and clears the popup window. The popup can also be closed by clicking on the tab for original page.

The popup windows shown in FIG. 19 are additional browser windows that are shown separate to the user's current view. A popup window may be a completely separate item (e.g. an advertisement) or it may have some relationship to the page that caused it to pop up (e.g. it might contain a calendar used for selecting a date that is then automatically copied back to the original page). A popup window may be closed automatically or it may require the user to explicitly close it.

Typical desktop Operating Systems allow multiple windows to show at the same time making it easy for the user to see and navigate between multiple windows. On a small screen device such as a PDA or phone, however, it is common to show only a single window at a time. This could be because of restrictions in the Operating System or because of constraints in the size of the screen.

In one embodiment, the popup window might be inserted into the screen history list of the browser. This can allow the user to easily navigate between the original window and any popup windows generated from it. Further, by retaining the internal representation of the windows in screen history in memory, interactions between the pages, such as that displayed in FIG. 19, can be completed without the need for actual

separate windows. In this embodiment, the act of closing a popup window might be equivalent to going back in screen history.

In another embodiment, the invention displays popup windows as separate views stacked on top of each other, exposing only a tab that can be used to select the current one. When the tab for a window is selected, the view of that window will be brought to the front. Similarly, if a window is brought to the front (e.g. by clicking on a link in another window), its tab will be selected. When a window is closed (either explicitly by the user or automatically by a script command), then it and its tab are removed. The main browser window occupies the left most tab and cannot be removed or closed.

In another embodiment shown in FIG 20, the full title of each page might be used to display more information about each available window. Typically a page can be identified by its title that on a desktop browser is often shown at the top of the window. In a typical PDA or browser environment where there are no windows, the invention displays this information in a separate area or status bar. Preferably, this status bar can include a control that allows the user to choose which "window" should currently be displayed in the main screen. The control may take the form of a popup select list, a "calendar roller" or other means of selecting from the list of pages. The list of pages can include those requested by URL, popup windows initiated by a user action or automatic popups (such as advertisement windows). As shown in FIG 20, one advantage of this method is that the user has more information available for selecting which page because the title is typically longer than can fit into a small tab. Preferably, popup pages that have

no title can be identified by the user action or content used to create them (e.g. "OnOpen popup" or "Popup Calendar Click").

Additionally, a user may wish to see the entire contents of a Web site without utilizing the expandable object architecture. Therefore, the user can support a "Load All" of a Web site, which can result in the full content tree of the Web site being sent to the device. In this mode, there will be no expandable objects, however the normalization can utilize white-space horizontal tabs to display the parent-child relationship.

The server browser can have a minimum character threshold used to determine if the children under a parent are small enough to send on a load. The server browser preferably has a maximum children threshold used to determine if the number of children under a parent is small enough to send on a load. If the maximum character threshold is not exceeded and the maximum children threshold is not exceeded, the children will be sent in the load. The server browser preferably has an inline threshold expansion parameter in which horizontal tabs will be used to show the parent-child relationship rather than expandable objects.

In addition, the user can utilize a find function on the electronic device. The find function can send a request to the server browser for a find to be performed on the page with a user input string. Options for loading a page and their effect on the content is specified in the following table.

	Threshold Exceeded	**Threshold Not Exceeded/Threshold Inline On**	**Threshold Not Exceeded/Threshold Inline Off**
Load All/ No Tables	Thresholding has no effect. All content will be sent formatted inline.	Thresholding has no effect. All content will be sent formatted inline.	Thresholding has no effect. All content will be sent formatted inline.

	Threshold Exceeded	Threshold Not Exceeded/Threshold Inline On	Threshold Not Exceeded/Threshold Inline Off
L ad Summary/ No Tables	Normalized content will be sent formatted as expandable objects.	Children under the threshold will be sent formatted inline.	Children under the threshold will be sent formatted as expandable objects.
Load All/ with Tables	Thresholding has no effect. All content will be sent formatted Inline except for tables.	Thresholding has no effect. All content will be sent formatted Inline except for tables.	Thresholding has no effect. All content will be sent formatted Inline except for tables.
Load Summary/ with Tables	Normalized content will be sent formatted as expandable objects, except for tables.	Children under the threshold will be sent formatted Inline. Tables will be sent as is.	Children under the threshold will be sent formatted as expandable objects. Tables will be sent as is.

The server browser preferably employs methods to partition content based on user profile characteristics. These characteristics could be keywords or usage statistics. The data content sent to the electronic device may need to be adapted to a specific markup language or device dependent requirements (e.g. display size, color capability etc). The protocol and data content sent to the electronic device is preferably compressed and optimized to reduce bandwidth usage and cost to the user.

User authentication and secure communication is preferred by certain web sites for private end-to-end communication (from the Internet to the electronic device). This secure communication and authentication is accomplished by many mechanisms such as, SSL and HTTP Authentication. HTTP Basic Access Authentication scheme is a method of user authentication, by passing a user name and password back to a web server. Basic Authentication does not encrypt the username and password back to the web server.

The server browser preferably supports HTTP 1.1 Basic Access Authentication. The Device shall have the ability to allow the user to enter the username and password

for authentication. The SSL Protocol preferably provides privacy and reliability between at least two electronic devices. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP, is the SSL Record Protocol.

The SSL Record Protocol can be used for encapsulation of various higher level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent. A higher level protocol can layer on top of the SSL Protocol Transparently." The Transport Layer Security ("TLS") protocol is a successor to SSLv3. It is an IETF RFC rather than a defacto industry standard. The server browser preferably supports SSL version 3.0 or later.

In another aspect, the client browser can optionally utilize the server browser as a means to enhance capabilities, improve speed or add function. The use of the server browser by the client browser can be initiated either manually via a user preference or automatically via pre-defined algorithms that take into account the hardware and software capabilities of the device and the characteristics of the wireless network used for the request/response as well as the application needs indicated by the information source. Preferably, the client browser can automatically recognize conditions in which the server browser is either not needed or causes problems accessing the information source.

One such example is the use of authentication by WiFi (802.11) hotspots. The hotspots available today (e.g. in Starbucks, MacDonalds, etc.) commonly require the user to manually disable all proxy servers in order to log in and authenticate on the network. In one embodiment, browser or other applications might automatically switch to proxyless mode for the authentication and then switch back to server mode once authentication is complete. FIG. 29 shows an exemplary process for deciding when to automatically switch the client browser from server mode to proxyless mode and back again.

In another aspect, the client examines the characteristics of the underlying network connection to determine when to operate in server or proxyless mode. This is of benefit for those devices that can support multiple types of networks through either built in hardware or after market add-ons. The market already contains devices with these capabilities and more are expected. For example, Palm devices are available with both built in GPRS radio and Secure Digital expansion slot that can support a WiFi (802.11) card. FIG. 30 shows an exemplary process for determining the mode automatically based on user settings. The decision for which mode to switch to is based on criteria specified by the user, including such aspects as:

- a pre-set mode associated with particular network selection
- determining optimum performance based on particular network characteristic, such as speed, bandwidth, etc.
- determining optimum cost based on particular network characteristics such as connection cost, data transfer costs, etc.

It can be seen from FIG. 30 that the mode might not immediately be changed when the underlying network changes. This allows the server component to maintain the same session across networks and network changes even when the mode changes.

The present embodiments, described herein as exemplary embodiments, provide a unique approach to mobilizing critical enterprise content and applications without customizations. By transforming and adapting data content in real-time, dynamic information that is organized particularly for small-screened, wireless devices can be delivered. The present embodiments optimize and organize data content such as Internet content, preferably eliminating the need for custom programming or "mirrored" Web sites in non-traditional markup languages such as WML, HDML, C-HTML or XML. Markup languages such as HTML, JavaScript and WML content can be read and transform the existing content into either XML or WML to adapt to the ergonomics of handheld devices, so no additional development costs are incurred.

It should be understood that the programs, processes, methods and systems described herein are not related or limited to any particular type of computer or network system (hardware or software), unless indicated otherwise. Various types of general purpose or specialized computer systems may be used with or perform operations in accordance with the teachings described herein.

In view of the wide variety of embodiments to which the principles of the present invention can be applied, it should be understood that the illustrated embodiments are exemplary only, and should not be taken as limiting the scope of the present invention. For example, the steps of the flow diagrams may be taken in sequences other than those

described, and more or fewer elements may be used in the block diagrams. While various elements of the preferred embodiments have been described as being implemented in software, in other embodiments in hardware or firmware implementations may alternatively be used, and vice-versa.

It will be apparent to those of ordinary skill in the art that methods involved in the system and method for compressing and decompressing a binary code image may be embodied in a computer program product that includes a computer usable medium. For example, such as, a computer usable medium can include a readable memory device, such as a hard drive device, CD-ROM, a DVD-ROM, or a computer diskette, having computer readable program code segments stored thereon. The computer readable medium can also include a communications or transmission medium, such as, a bus or a communication link, either optical, wired or wireless having program code segments carried thereon as digital or analog data signals.

The claims should not be read as limited to the described order or elements unless stated to that effect. Therefore, all embodiments that come within the scope and spirit of the following claims and equivalents thereto are claimed as the invention.